

FLAM

FRANKENSTEIN-LIMES-ACCESS-METHOD

für UNIX

BENUTZERHANDBUCH

— Ausgabe März 2014 Version 4.3 —

Telefon (06172) 5919-0 ■ Telefax (06172) 5919-39
http://www.flam.de

Benutzerhandbuch FLAM V4.3 (UNIX)

FLAM (UNIX)

Benutzerhandbuch

Inhaltsverzeichnis

Inhalt

FLAM (UNIX)

Benutzerhandbuch

Kapitel 1:

FLAM im Überblick

1. FLAM im Überblick

FLAM ist eine Realisierung der FLAM-Algorithmen für Rechner unter dem Betriebssystem UNIX. Es dient zur Komprimierung und Dekomprimierung von Dateien in einem Format, das den problemlosen Datenaustausch auch mit zahlreichen Rechnern anderer Hersteller ermöglicht. Darüber hinaus bietet FLAM auch die Möglichkeit, Daten komprimiert oder unkomprimiert zu verschlüsseln

Wegen der heterogenen Kompatibilität eignet FLAM sich auch insbesondere zur langfristigen Aufbewahrung von Daten, da die Reproduzierbarkeit der Daten nicht mehr an den Lebenszyklus bestimmter Systeme gebunden ist. Ferner können mit Hilfe von FLAM Dateien in fast beliebiger Weise konvertiert, d.h. in Dateien mit einer anderen Organisation, einem anderen Satzformat oder einem anderen Zeichencode überführt werden.

Die Architektur dieser Software ist schichtenorientiert, d.h. sie setzt sich aus mehreren hierarchisch geordneten Funktionsgruppen zusammen, die klar gegeneinander abgegrenzt sind und über definierte Schnittstellen miteinander kommunizieren. Durch Offenlegung dieser Schnittstellen ist es dem Benutzer überlassen, selbst zu entscheiden, wie weit er FLAM in seine Anwendungen integriert.

FLAM kann sowohl per Kommando als auch über die entsprechenden Schnittstellen von Anwenderprogrammen aus aufgerufen werden. Obwohl FLAM beliebige Dateitypen komprimieren kann, ist es ursprünglich konzipiert für Dateien mit weitgehend einheitlicher Satzstruktur und zeichencodierten Daten. In diesem Bereich erzielt es eine besonders hohe Effizienz. Die genannten Voraussetzungen sind meistens gegeben, wenn die verarbeitenden Anwenderprogramme in einer höheren Programmiersprache geschrieben sind. Dieses Kapitel soll dem Benutzer bei der Entscheidung helfen, wann und wie FLAM sinnvoll und mit größtmöglichem Nutzen eingesetzt werden kann.

Beim Kommando flam können die erforderlichen Angaben als Parameter in der Kommandozeile oder als FLAM-Parameter in einer Parameterdatei enthalten sein, auf die mit dem Parameter -parfile=parameterfile verwiesen wird. In Kapitel 2, "Das Kommando flam", wird hierauf näher eingegangen. Sofern auf Unterschiede nicht ausdrücklich hingewiesen wird, gilt für dieses Kapitel generell, dass alle Erläuterungen über die Verwendung der Parameter im Kommando gleich der in der Parameterdatei sind, ohne dass dies einer besonderen Erwähnung bedarf.

1.1 Arbeitsweise

Bei der Komprimierung einer Datei mit FLAM wird eine oder mehrere Originaldatei(en) satzweise gelesen und eine (ggf. verschlüsselte) Komprimatsdatei erzeugt, die sog. FLAMFILE . Bei der Dekomprimierung einer Datei mit FLAM wird die Komprimatsdatei sequentiell gelesen und eine oder mehrere dekomprimierte Datei(en) erzeugt.

Der Komprimierungsprozess besteht aus sich wiederholenden Zyklen. Bei jedem Zyklus wird in der Regel eine bestimmte Anzahl von Sätzen der Originaldatei gelesen, in einem Zwischenpuffer, dem Matrixpuffer, zwischengespeichert, komprimiert und in komprimierter Form als ein Komprimatsblock ausgegeben. Diese Anzahl kann je nach verwendetem Verfahren zwischen 1 und 255 bzw. 4095 liegen und wird beim Komprimierungsaufruf vorgegeben. Bei Erreichen des Dateiendes oder bei Überlauf des Matrixpuffers wird ein Komprimatsblock mit entsprechend weniger Sätzen erzeugt.

Die von FLAM erzeugten Komprimatsblöcke werden in eine sequentielle FLAMFILE geschrieben.

Durch das satzweise Lesen der Originaldatei bleibt die Strukturinformation der Datei erhalten und gestattet beim Dekomprimieren insbesondere in Rechnerumgebungen mit Datenverwaltungssystemen eine strukturgetreue Reproduktion der Originaldatei.

Jeder Satz der FLAMFILE wird je nach Komprimierungsmodus mit einer binären Prüfsumme bzw. einem Prüfzeichen versehen, die bei der Dekomprimierung zur Verifikation der Datenintegrität überprüft wird. Die FLAMFILE kann gespeichert oder mit anderen Rechnern ausgetauscht werden. Bei der Dekomprimierung kann die dekomprimierte Datei - eventuell abweichend von der Originaldatei - entsprechend den Vorgaben des Benutzers oder der Default-Einstellungen mit einer anderen Organisation, einem anderen Satzformat oder mit veränderter Satzlänge erzeugt werden.

1.2 FLAM-Modi

FLAM kennt vier Komprimierungsmodi: **adc**, **cx7**, **cx8**, **vr8**. Im Modus adc, cx8 und vr8 erzeugt FLAM die FLAMFILE als binäre Datei.

adc ist der effizienteste Komprimierungsmodus. Er komprimiert in Richtung des Datenflusses durch Eliminierung von Redundanz sich wiederholender Bytefolgen und eignet sich zur Anwendung auf beliebige Daten. Die anderen Modi reduzieren vorwiegend "vertikale Redundanz", d.h. ihre Wirksamkeit beruht auf der Erkennung gleicher Spalteninhalte bei tabellarisch strukturierten Daten.
Der Komprimierungsmodus adc kann mit Verschlüsselung kombiniert werden, wofür FLAM zwei Verfahren anbietet, nämlich das im Jahr 2002 vom U.S.-Normeninstitut NIST zum Standard erklärte AES und ein proprietäres Verfahren der limes datentechnik gmbh. Diesen beiden möglichen Kombinationen entsprechen die FLAM-Modi **aes** bzw. **flamenc**. Mit den anderen Komprimierungsmodi ist eine Verschlüsselung nicht möglich.

Bei den binären Modi ist die komprimierte Codierung der Information weitgehend unabhängig von Charakteristika des Systems, auf dem die Komprimierung ausgeführt wird, d.h. die FLAMFILEs dürfen nicht verändert werden, etwa durch Übertragung über Leitungen, die automatisch eine Codeübersetzung durchführen.

Im Modus cx7 komprimierte FLAMFILEs hingegen sind gegen Codeübersetzungen unempfindlich, da in ihnen alle Steuerinformationen durch alphanumerische Zeichen dargestellt werden, deren Codierungen sowohl im ASCII- als auch im EBCDIC-Code international standardisiert sind. Sofern die Originaldaten zeichencodiert sind und sich für einen Datenaustausch zwischen Systemen mit unterschiedlichen Zeichencodes eignen, bleibt der Informationsgehalt der FLAMFILE nach einer Übersetzung in den Zeichencode des Zielsystems trotz des geänderten binären Inhalts dank der zeichenweisen Interpretation vollständig erhalten.

Bei der Dekomprimierung wird der Komprimierungsmodus automatisch erkannt. Der Benutzer braucht daher den Komprimierungsmodus nicht zu kennen, um eine Datei zu dekomprimieren.

1.3 Matrixpuffer und Satzanzahl

Die Größe des Matrixpuffers und die Anzahl der dort gepufferten Sätze können bei der Komprimierung mit den Parametern maxbuffer (nicht bei Modus adc) und maxrecords vorgegeben werden. Als Faustregel gilt, dass die Komprimierung um so effizienter ist, je mehr Sätze in einem Komprimatsblock zusammen komprimiert werden. Wenn nicht andere Gesichtspunkte dem entgegenstehen, empfiehlt es sich i.a., das Maximum (255 bzw. 4095) zu wählen. Die vorgegebene Satzanzahl stellt allerdings lediglich eine Obergrenze dar. Sie kann unterschritten werden, wenn der Platz im Matrixpuffer erschöpft ist oder wenn das Ende der Originaldatei erreicht ist.

Welche Größe des Matrixpuffers benötigt wird, hängt ab von der Anzahl der Sätze, die gepuffert werden sollen, und deren Längen. Bei Dateien mit Sätzen fixer Länge errechnet sich der Platzbedarf einfach als Produkt von Satzanzahl und -länge. Bei Dateien mit variablen Satzlängen ist eine exakte Voraussage des Speicherbedarfs i.a. nicht möglich. Daher kann nur ein Näherungswert gewählt werden. Es muss jedoch zumindest gewährleistet sein, dass der Matrixpuffer ausreicht, um jeden Satz der Datei einzeln zu fassen.

1.4 Die FLAMFILE

1.4.1 Aufbau der FLAMFILE

Um die universelle Austauschbarkeit zu gewährleisten, muss die FLAMFILE in einem Format aufgebaut sein, das auf allen Betriebssystemen darstellbar ist. Dieser "kleinste gemeinsame Nenner" ist die sequentielle Datei mit Sätzen fixer Länge. Zwar sind für die FLAMFILE neben Satzformat fix auch weitere Satzformate zulässig, doch wird das bei der Komprimierung erzeugte Komprimat in "abstrakte" Sätze, die sog. FLAM-Sätze gleicher Länge zerlegt und diese auf "konkrete" Strukturen des jeweiligen Betriebssystems abgebildet. Jeder FLAM-Satz enthält neben den Komprimatsdaten am Anfang ein FLAM-Satzlängenfeld und am Ende eine Prüfsumme oder ein Prüfzeichen. Das FLAM-Satzlängenfeld und die Prüfsumme bzw. das Prüfzeichen sind Bestandteile der FLAM-Syntax. Diese ermöglicht die Erkennung von Anfang und Ende eines FLAM-Satzes unabhängig vom Satzformat, in dem das Betriebssystem die FLAMFILE speichert.

Bei FLAM ist ein FLAM-Satz identisch mit einem Satz, wenn die FLAMFILE fixes und variables Satzformat hat. Die vorgegebene FLAM-Satzlänge ist dann die Satzlänge bzw. maximale Satzlänge.

Bei FLAMFILEs werden die Restdaten eines Komprimatsblocks mit Daten des Folgeblocks zu einem

FLAM-Satz der fixen bzw. maximalen Länge zusammengefügt. Die FLAM-Sätze, bis evtl. auf den letzten, sind alle gleich lang. Dadurch wird die Nutzung des Speichermediums optimiert, doch ist es dabei nicht möglich, Sätze der Originaldatei und Sätze einer FLAMFILE einander eindeutig zuzuordnen. Eine solche FLAMFILE kann deshalb immer nur komplett dekomprimiert werden.

Bei FLAM sind für die FLAMFILE die Satzformate fix und variable bzw. stream zulässig. Unabhängig vom Satzformat ist die kleinste zulässige Länge für den FLAM-Satz 80 und die größtmögliche 32.760 Bytes (bei -mode=cx7 4095), sofern keine niedrigere Obergrenze gesetzt wird. Diese Spanne reicht praktisch aus, um die Übertragbarkeit einer FLAMFILE selbst da zu gewährleisten, wo dies für die Originaldatei aufgrund von Restriktionen des eingesetzten File-Transfer-Produkts bezüglich der Satzlängen oder der Satzformate nicht ohne weiteres möglich ist.

Wenn sich auch viele Probleme beim Datenaustausch durch das FLAM-Konzept entschärfen lassen, können gewiss nicht alle dadurch gelöst werden. Eignet sich eine Datei etwa wegen der darin enthaltenen Binärdaten nicht zur unkomprimierten Übertragung mit einem bestimmten File-Transfer-Produkt, so wird auch nach der Komprimierung mit FLAM eine Übertragung nicht möglich sein, selbst bei einer Komprimierung im Modus cx7. Denn prinzipiell kommt jede im Original vorkommende Bitkombination auch in der FLAMFILE vor.

1.4.2 Die FLAM-Fileheader

Neben dem Datenkomprimat kann eine FLAMFILE weitere Informationen enthalten, die in den FLAM-Fileheadern gespeichert werden. FLAM unterscheidet zwei Kategorien von Informationen, für die jeweils ein eigener Fileheader-Typ vorgesehen ist:

- Allgemeine Informationen werden im allgemeinen FLAM-Fileheader abgelegt. Dies sind Angaben über die Originaldatei, wie Dateiorganisation, Satzformat, Länge und Position des Primärschlüssels etc., aber auch Informationen über die FLAM-Version und das Betriebssystem, unter dem die FLAMFILE erzeugt wurde.

- Für anwenderspezifische Informationen beliebiger Art gibt es den anwenderspezifischen FLAM-Fileheader. Dieser kann bei Benutzung der Satzschnittstelle zusätzlich eingefügt werden, etwa für Informationen, die vom Anwenderprogramm auszuwerten sind.

Die Erzeugung des allgemeinen FLAM-Fileheaders wird von allen Schnittstellen (Kommando, Unterprogramm- und Satzschnittstelle) durch entsprechende Parameter und Argumente unterstützt. Der anwenderspezifische FLAM-Fileheader kann ausschließlich bei Benutzung der Satzschnittstelle erzeugt werden. Diese Schnittstelle bietet folgende Funktionen für die Erzeugung und Auswertung der verschiedenen Fileheadertypen:

FLAM-Fileheadertyp	Erzeugung	Auswertung
Allgemein	flmphd	flmghd
Anwenderspezifisch	flmpuh	flmguh

Die Einfügung von FLAM-Fileheadern in die FLAMFILE ist nicht zwingend erforderlich. Werden sie eingefügt, ist bei den Funktionen der Satzschnittstelle auf die Einhaltung der vorgeschriebenen Reihenfolge zu achten (Siehe Näheres im Abschnitt 1.5.3).

FLAM-Fileheader stehen in der FLAMFILE stets unmittelbar vor dem Komprimat der zugehörigen Datei. Enthält eine FLAMFILE Komprimate mehrerer Dateien, so können vor jedem Komprimat entsprechende FLAM-Fileheader eingefügt werden.

Beim Dekomprimieren können so Komprimate verschiedener Dateien unterschieden und in getrennte Dateien dekomprimiert werden.

1.4.3 Secure FLAMFILEs

Generell ist jeder Satz einer FLAMFILE zum Schutz gegen Manipulationen oder Übertragungsfehler mit einer Prüfsumme weitgehend abgesichert. Im Einklang mit den hohen Anforderungen an die kryptographische Sicherheit wurden für verschlüsselte FLAMFILEs weitere Sicherheitsmerkmale implementiert, die die Daten auch gegen Angriffe anderer Art besser schützen. Mit diesen Sicherheitsmerkmalen versehene FLAMFILEs heissen "Secure FLAMFILEs".

Die neuen Schutzmechanismen wirken auf drei Ebenen:
- der Segment-Ebene,
- der Member-Ebene und
- der File-Ebene.

Dabei bezeichnet "Segment" das Komprimat einer einzelnen Matrix und "Member" die Gesamtheit der

Segmente, die das Komprimat einer Originaldatei enthalten.

In einer Secure FLAMFILE werden sowohl für jedes Member als auch für die FLAMFILE insgesamt Zähler für die unkomprimierten und die komprimierten Bytes und Sätze, fortlaufende Nummern und ein Zeitstempel mitgeführt. Bei Verschlüsselung mit AES werden so genannte MACs (Message Authentication Codes) berechnet. Dies sind mit Schlüsseln versehene Prüfsummen, die zur Verifizierung der Authentizität der Daten dienen. Für jedes Segment wird ein solcher Segment-MAC ermittelt. Ein weiterer MAC, der mit den entsprechenden MACs der vorangehenden Segmente verkettet ist, sichert die Vollständigkeit der Segmente im Member ab. Auf gleiche Weise wird durch eine weitere MAC-Kette die Vollständigkeit der Member im File sichergestellt. Ähnliche Merkmale haben auch FLAMFILEs, die mit mode=flamenc erzeugt werden.

Zur Aufnahme der für diese Prüfungen benötigten Informationen wie Zähler, Zeitstempel, MACs etc. wird vor jedem Member ein Memberheader geschrieben und dahinter ein Membertrailer. Zusätzlich wird ans Ende einer Secure FLAMFILE ein Filetrailer angehängt.

Secure FLAMFILEs können nicht konkatiniert werden, da dadurch Unstetigkeiten in der Nummerierung, den Zählern und der MAC-Verkettung entstehen, die von FLAM als Integritätsverletzung gedeutet werden und zum Fehlerabbruch führen. Hingegen können einzelne Member einer Secure FLAMFILE dekompromiert werden, da FLAM beim Überspringen von Membern innerhalb einer FLAMFILE die MACs der Memberverkettung nicht mehr prüft, sondern lediglich die Segmentverkettung verifiziert.

1.5 Schnittstellen

Die verschiedenen Schnittstellen, die FLAM besitzt, gestatten eine vielfältige Verwendungsweise.

Durch Aufrufe aus der entsprechenden Ebene kann FLAM in eigene Anwendungen oder diese in FLAM eingebettet werden. Die folgende Tabelle gibt einen Überblick über die vorhandenen Schnittstellen:

Schnittstelle	Aufrufebene	Zweck
Kommando flam	System	Komprimierung/ Dekomprimierung ganzer Dateien im interaktiven Betrieb oder in Prozeduren
Unterpro-gramm flamup	Anwendungs-programme	Komprimierung/ Dekomprimierung ganzer Dateien in eigenen Anwendungen
Satzschnitt-stelle flamrec	Anwendungs-programme	Übergabe und Abruf einzelner Sätze in eigenen Anwendungen
Benutzer-eigene Ein-/Ausgabe	FLAM	Ersatz der FLAM-Ein- und Ausgaberoutinen
Benutzer-ausgänge	FLAM	Vor- bzw. Nachschal-tung eigener Datei- und/oder Satz-verarbeitung für komprimierte und unkomprimierte Daten; Schlüsselverwaltung

Diese Schnittstellen besitzen eine hierarchische Ordnung, die festlegt, welche Schnittstelle von wo aus benutzt bzw. aufgerufen werden kann. Die Schnittstellen unterteilen sich in zwei Kategorien: aktive und passive Schnittstellen. Der Aufruf aktiver Schnittstellen wird vom Benutzer bzw. dessen Anwenderprogramm initiiert, und aktiviert die sie unterstützenden Routinen von FLAM. Hierzu zählen das Kommando, die Unterprogramm- und die Satzschnittstelle. Über passive Schnittstellen werden von FLAM Benutzerroutinen aktiviert, die einen bestimmten Funktionsumfang erfüllen müssen, deren Aufruf aber nicht mit der Logik des Anwenderprogramms synchronisiert ist. In diese Kategorie fallen die benutzereigene Ein-/Ausgabe und die Benutzerausgänge.

Die folgenden Abschnitte geben einen Überblick, wie die verschiedenen Schnittstellen zu benutzen sind, ohne auf alle Details einzugehen. Für jede Schnittstelle enthält eines der Folgekapitel eine ausführliche Beschreibung, die als Referenzdokument dient. Alle Programmschnittstellen, d.h. alle Schnittstellen mit Ausnahme des Kommandos, sind im jeweiligen Referenzkapitel in C-Notation dargestellt, und für C-Programmierer ist die C-Header-Datei flamincl.h verfügbar, in der die C-Konstantendefinitionen enthalten sind. Doch unterstützen diese Schnittstellen keineswegs nur C-Programme. Vielmehr können sie mit beliebigen Programmiersprachen genutzt werden, da generell anstelle der C-typischen Argumentübergabe als Werte alle Argumente durch Pointer, also durch ihre Adressen übergeben werden.

Wenn auch bei dieser Form der Übergabe der Name ohne den "*"-Operator im strengen C-Sinne die Argumentadresse kennzeichnet, sind bei Bezugnahmen auf diese Argumente in den nachstehenden Erläuterungen stets die Werte der Argumente gemeint.

1.5.1 Das Kommando flam

Das Kommando flam kann benutzt werden, um interaktiv oder in Prozeduren einzelne Dateien oder Gruppen von Dateien zu komprimieren oder zu dekomprimieren. Die Parameter compress bzw. decompress geben hierbei an, welche Operation ausgeführt werden soll. Alle zur Ausführung des Kommandos benötigten Angaben können durch Eingaben in der Kommandozeile selbst, über eine Parameterdatei oder über installationsspezifische Defaultwerte eingestellt werden.

Das flam-Kommando mit dem Parameter list zeigt die aktuellen Einstellungen dieser Defaultwerte an, und mit dem Parameter defaults können sie vom Systemmanager geändert werden.

Mit dem flam-Kommando kann aus je einer Originaldatei eine einzelne FLAMFILE erzeugt werden. Es können auch mehrere Originaldateien in eine einzige FLAMFILE komprimiert werden.

Bei der Komprimierung können die Eingabedateien explizit benannt oder implizit durch Suchmuster mit Wildcards spezifiziert werden, während für die Ausgabedateien neben expliziter Nennung auch Substitutionsvorschriften für die Namensbildung zulässig sind.

Dieselben Spezifikationsformen sind auch bei der Dekomprimierung möglich, doch kann zusätzlich durch die spezielle Ausgabespezifikation [ ] bzw. [*] eine Datei unter ihrem ursprünglichen Namen und mit ihren Originalattributen wiedererzeugt werden.

Dateiattribute der dekomprimierten Datei, wie Organisation oder Satzformat, können von denen der Originaldatei abweichen.

Dadurch kann FLAM auch zur Konvertierung von Dateien verwendet werden.

Mit Ausnahme der Parameter list und defaults gibt es zu allen Elementen der Kommandosyntax äquivalente FLAM-Parameter, so dass jede Einstellung sowohl durch Eingaben in der Kommandozeile als auch mittels Angaben in einer Parameterdatei möglich ist, auf die im Kommando mit dem Parameter -parfile=Parameterdatei verwiesen wird. Durch Benutzung einer Parameterdatei können z.B. umfangreiche Tastatureingaben vermieden werden, wenn Einstellungen, die von den installierten Defaultwerten abweichen, häufig benötigt werden.

1.5.2 Das Unterprogramm flamup

Mit der Schnittstelle zum Unterprogramm flamup können etwa dieselben Operationen, wie mit dem Kommando flam, aus einem Anwenderprogramm heraus ausgeführt werden.

Diese Schnittstelle stellt dem Anwendungsprogrammierer denselben Funktionsumfang zur Verfügung wie dem Benutzer, ausgenommen die Anzeige und Änderung von Defaulteinstellungen. Für die erforderlichen Einstellungen werden - ähnlich wie bei der Parameterdatei - FLAM-Parameter verwendet. Diese stehen hintereinander in einer ASCII-Zeichenkette, die dem Unterprogramm als Argument übergeben wird. Auch hier ist mit dem Parameter -parfile=Parameterdatei der Verweis auf eine Parameterdatei möglich. Für fehlende Angaben werden dieselben Defaulteinstellungen verwendet wie beim Kommando flam.

Nach Beendigung der Ausführung von flamup erhält das aufrufende Programm einen Returncode, der den Erfolg des Aufrufs bzw. die Fehlerursache anzeigt.

1.5.3 Die Satzschnittstelle flamrec

Mit Hilfe der Satzschnittstelle flamrec können Dateiverarbeitung und -komprimierung innerhalb eines Anwendungsprogramms integriert werden. Dekomprimierte Sätze können aus einer FLAMFILE sequentiell gelesen werden. Bei der komprimierten Ausgabe von Daten wird eine FLAMFILE neu erzeugt. Eine FLAMFILE kann also entweder nur für Lesezugriffe (Dekomprimierung) oder nur für Schreibzugriffe (Komprimierung) geöffnet werden.

Die Satzschnittstelle von FLAM besteht aus einer Klasse von Funktionen, die es dem Anwenderprogramm u.a. gestatten, auf komprimierte Daten so zuzugreifen und sie so zu verarbeiten, wie dies mit unkomprimierten Daten durch gewöhnliche Ein-/Ausgabebefehle geschieht. Diese Funktionen sind so konzipiert, dass ihre Benutzung ähnlichen Regeln unterliegt, wie sie für die Standard-Ein-/Ausgabe vorgegeben sind. Nach dem Öffnen der Datei können Sätze gelesen oder geschrieben werden, und am Ende der Verarbeitung wird die Datei geschlossen.

Die im Zuge der Verarbeitung stattfindende Komprimierung bzw. Dekomprimierung bleibt für den Programmierer unsichtbar, so dass die Benutzung der Satzschnittstelle gegenüber der Programmierung mit herkömmlichen Ein-/Ausgabebefehlen für den Programmablauf kaum Unterschiede in der Programmlogik beinhaltet.

Durch Benutzung der Satzschnittstelle entfällt die Notwendigkeit, die Komprimierung und Dekomprimierung der zu verarbeitenden Dateien als gesonderte Schritte nach bzw. vor der Ausführung des Anwendungsprogramms auszuführen. Auch ist es nicht mehr erforderlich, für den temporär erhöhten Speicherplatzbedarf auf dem Massenspeicher Kapazitäten vorzuhalten, um bis zum Abschluss der Komprimierung bzw. Dekomprimierung die komprimierte und die nicht komprimierte Version der Datei aufnehmen zu können.

1.5.4 Die benutzereigene Ein-/Ausgabe

FLAM unterstützt standardmäßig Ein- und Ausgabe auf Plattendateien und verwendet hierzu Standard-Ein-/ Ausgabebefehle. Es gestattet dem Anwender jedoch, die Standardroutinen für Ein- und Ausgabe durch eigene Routinen zu ersetzen. Dies kann z.B. dann sinnvoll sein, wenn ein anderes Ein-/Ausgabegerät als Platte in Betracht kommt oder die Datei ein nichtunterstütztes Satzformat enthält.

Es ist nicht möglich, die Aufgabenstellung einer jeden Funktion der benutzereigenen Ein-/Ausgabe im Detail zu spezifizieren, da sich diese je nach Gerät bzw. Art der

Daten anders darstellt. Beipielsweise erfordern Daten, die über eine DFÜ-Leitung empfangen werden, andere Aktionen als etwa Daten aus Messinstrumenten. FLAM betrachtet den Ursprung der Daten als "Datei", auch wenn es sich hierbei im konkreten Fall um andere Objekte handeln kann.

Die benutzereigene Ein-/Ausgabe umfasst folgende Funktionen:

Funktion	Zweck	Entspricht I/O Aufrufen in C
usrcls	Datei schließen	fclose
usrget	Einen Satz lesen	fgets
usropn	Datei öffnen	fopen
usrpos	Auf Satz positionieren	fseek
usrput	Einen Satz schreiben	fputs

Entsprechend dieser Sichtweise gibt FLAM für diese Schnittstelle einen an den Bedürfnissen der Dateiverarbeitung orientierten Funktionsumfang vor. Es ist die Aufgabe des Anwenders, der diese Schnittstelle nutzen will, die Funktionen so zweckgerecht zu konzipieren und auszugestalten, dass das angestrebte Resultat mit diesem Funktionsvorrat erreicht werden kann.

Um benutzereigene Ein-/Ausgabe verwenden zu können, müssen die entsprechenden Routinen mit dem Anwenderprogramm zusammengebunden werden. Deshalb wird diese Möglichkeit nur von den Programmschnittstellen aus unterstützt, d.h. von der Unterprogramm- und der Satzschnittstelle.

Die Unterstützung durch die Satzschnittstelle berührt natürlich nur Zugriffe auf die FLAMFILE, denn nur solche Zugriffe werden durch diese Schnittstelle ausgeführt.

Benutzereigene Ein-/Ausgabe wird durch Aufruf von flmopd mit -device=7 initiiert. Bei der Unterprogrammschnittstelle kann über die FLAM-Parameter user_io, inuser_io und outuser_io die benutzereigene Ein-/Ausgabe für alle bei der Komprimierung bzw. Dekomprimierung angesprochenen Dateien aktiviert werden.

Während bei den Funktionen der Satzschnittstelle die Beachtung gewisser Regeln hinsichtlich der Reihenfolge und Konsistenz der Aufrufe dem Programmierer obliegt, wird
das richtige Zusammenspiel der Funktionen der benutzereigenen Ein-/Ausgabe durch FLAM gewährleistet, bei dem die Initiative für die Aufrufe liegt.

Der Programmierer der benutzereigenen Ein-/Ausgabe-Routinen ist lediglich für die Korrektheit der Einzelfunktionen hinsichtlich ihrer Aktionen und Verhaltensweisen verantwortlich. Die Beschreibung in diesem Überblick beschränkt sich daher auf diese Aspekte.

Für jede zugeordnete Datei wird usropn als erste genau einmal aufgerufen. Mit dem Argument workio wird ein Arbeitsbereich von 1024 Bytes als dateispezifisches Gedächtnis zur Verfügung gestellt. Dieser Bereich wird bei allen nachfolgenden Aufrufen bis zum usrcls unverändert weitergegeben.

Im Argument openmode wird die Art des gewünschten Zugriffs (input, output) spezifiziert. In den Argumenten record_format, record_size usw. werden die Datei- und Satzattribute spezifiziert, die ggf. an die Gegebenheiten der Datei angepasst werden können.

Über fest definierte und frei vergebbare Returncodes können der erfolgreiche Abschluss der Funktion, bzw. spezielle Zustände und Fehler gemeldet werden. Der Returncode wird von FLAM ausgewertet und im Falle eines Fehlers ggf. an aufrufende Programme weitergeleitet.

Mit usrput wird ein Satz zum Schreiben übergeben. Kann der Satz nicht in der angegebenen Länge geschrieben werden, ist die Verkürzung oder die Auffüllung mit dem beim usropn in padchar angegebenen Füllzeichen im Returncode zu melden.

Mit usrget fordert FLAM den nächsten Satz an. Es dürfen maximal so viele Zeichen übergeben werden, wie im Parameter buffer_length angegeben sind. Muss der Satz deshalb verkürzt werden, ist das im Returncode zu melden.

Wird das Dateiende erreicht, ist das ebenfalls im Returncode zurückzumelden. Für jeden gelesenen Satz ist die Satzlänge zurückzugeben (auch bei fixem Satzformat).

Mit usrcls wird das Schließen der Datei veranlasst. Der Arbeitsbereich für diese Datei wird von FLAM nach Rückgabe der Kontrolle wieder freigegeben.

urspos ist bei UNIX eine leere Funktion.

1.5.5 Die Benutzerausgänge

Ein Benutzerausgang ist ein Programm, das vom Benutzer zur Verfügung gestellt wird und von FLAM an bestimmten Punkten des Programmablaufs über eine hierfür definierte Schnittstelle aufgerufen wird.

FLAM unterstützt zwei Kategorien von Benutzerausgängen: Benutzerausgänge für Dateizugriffe und einen Benutzerausgang für automatische Schlüsselverwaltung.

1.5.5.1 Benutzerausgänge für Dateizugriffe

Diese Benutzerausgänge gestatten dem Anwender, die von FLAM ausgegebenen Sätze sowohl der FLAMFILE als auch der dekomprimierten Datei vor ihrer endgültigen Speicherung in beliebiger Weise nachzubearbeiten, d.h. sie zu verändern, sie zu löschen oder zusätzliche Sätze einzufügen. Auch die von FLAM gelesenen Sätze können in analoger Weise für die Verarbeitung durch FLAM vorbereitet werden.

Unter Zugriff ist in diesem Zusammenhang ein Ein-/Ausgabebefehl oder ein entsprechender Aufruf der benutzereigenen Ein-/Ausgabe zu verstehen und nicht etwa der Aufruf einer Funktion der Satzschnittstelle. Es besteht also zumindest für die FLAMFILE keine Synchronisation der Benutzerausgänge mit dem Anwenderprogramm, denn für das letztere ist z.B. nicht erkennbar, ob ein flmget-Aufruf tatsächlich eine oder gar mehrere Leseoperationen auslöst oder nicht.

Die Aktivierung dieser Ausgänge ist über die Unterprogrammschnittstellen und die Satzschnittstelle möglich, wobei die Satzschnittstelle nur Benutzerausgänge für FLAMFILE-Zugriffe aktivieren kann. Die über diese Schnittstellen aufgerufenen Programme müssen mit der Anwendung statisch gebunden sein (s. Abschnitt 6.2).

Ein Benutzerausgang wird beim Öffnen, bei jedem Lesen oder Schreiben und beim Schließen der Datei aufgerufen. Über das Argument functioncode wird dem Benutzerausgang angezeigt, um welche Art des Zugriffs es sich handelt, um diesem eine situationsgerechte Reaktion zu ermöglichen.

FLAM unterstützt folgende Ausgänge für Dateizugriffe:

Benutzerausgang	Aufgerufen bei Zugriffen auf
exk10	Originaldatei (Komprimierung)
exk20	FLAMFILE (Komprimierung)
exd10	Dekomprimierte Datei (Dekomprimierung)
exd20	FLAMFILE (Dekomprimierung)

Bei der Komprimierung können die Benutzerausgänge exk10 und exk20 für Zugriffe auf die Originaldatei bzw. auf die FLAMFILE benutzt werden, bei der Dekomprimierung exd10 und exd20 für Zugriffe auf die dekomprimierte Datei bzw. die FLAMFILE. Wurden Sätze der FLAMFILE bei der Komprimierung durch den Benutzerausgang exk20 modifiziert, so ist eine Dekomprimierung nur möglich, wenn dies durch den komplementären Benutzerausgang exd20 revidiert wird, d.h. jeder FLAM-Satz durch exd20 in den selben Zustand zurückversetzt wird, wie er von FLAM am Benutzerausgang exk20 übergeben wurde.

1.5.5.2 Benutzerausgang für automatische Schlüsselverwaltung

Automatische Schlüsselverwaltung erleichtert dem Benutzer den Umgang mit verschlüsselten Dateien, indem sie ihm die Aufgabe abnimmt, Schlüssel zu erzeugen, auszutauschen und zu speichern, in welcher Form auch immer.

Über die Schnittstelle zum Benutzerausgang für automatischen Schlüsselverwaltung übergibt FLAM beim Verschlüsseln ggf. eine vom Benutzer spezifizierte Zeichenkette an die Schlüsselverwaltung und erhält zurück das Kennwort, das dann zum Verschlüsseln benutzt wird. Daneben kann die Schlüsselverwaltung FLAM eine Bytefolge zurückgeben, die der verschlüsselten Datei angeheftet wird und bei der Entschlüsselung zur Auffindung des passenden Schlüssels der Schlüsselverwaltung übergeben wird.

Die Aktivierung dieses Ausgangs ist über das Kommando flam und die Unterprogrammschnittstelle möglich, nicht jedoch über die Satzschnittstelle. Die über diese Schnittstellen aufgerufenen Programme müssen als shareable objects erzeugt und mit der Anwendung dynamisch gebunden sein (s. Abschnitt 6.2).

Hinweis: In diesem Handbuch werden die Begriffe "Kennwort" und "(kryptographischer) Schlüssel" synonym verwendet.

FLAM (UNIX)

Benutzerhandbuch

Kapitel 2:

Das Kommando flam

2. Das Kommando flam

2.1 Funktionen

Mit dem Kommando flam können folgende Funktionen aufgerufen werden:

- Komprimierung von Dateien mit oder ohne Verschlüsselung (mit dem Parameter -compress)
- Dekomprimierung von Dateien ggf. mit Entschlüsselung (mit dem Parameter -decompress)
- Ändern von installationsspezifischen Standard-einstellungen (Defaultwerten) (mit dem Parameter -defaults)
- Anzeige von installationsspezifischen Standard-einstellungen (mit dem Parameter -list)

Beim Aufruf des flam-Kommandos kann jede der erforderlichen Angaben auf unterschiedlichen Spezifikationsebenen, nämlich explizit im Kommando selbst, über eine Parameterdatei oder implizit durch Defaultwerte spezifiziert werden. Die nachstehende Beschreibung der Kommando-Syntax stellt die logischen Beziehungen zwischen den Syntaxelementen, also den Parametern und deren Argumenten dar, unabhängig davon, auf welcher Ebene die Spezifikation tatsächlich erfolgt.

Die nachstehende Einstufung der Syntaxelemente als obligatorisch oder optional gilt für diese Prüfung durch FLAM. Die Syntaxregeln beziehen sich demnach auf die Gesamtheit der Einstellungen, nicht allein auf die Eingaben der Kommandozeile.

In der Syntaxbeschreibung kann daher die Spezifikationsebene, ausser für die Prioritätsregeln, unberücksichtigt bleiben. Soweit im folgenden also von Parametern die Rede ist, gelten die Aussagen unverändert, wenn die entsprechenden Angaben durch das flam-Kommando, die Parameterdatei oder Defaultwerte ersetzt werden. Auf Ausnahmen hiervon wird besonders hingewiesen.

Bei der Eingabe des Kommandos FLAM kann für FLAM, FLAMFILE und auch alle anderen Parameterbezeichnungen Groß- und Kleinschreibung benutzt werden.

2.2 Kommando-Syntax

Mit dem Shellkommando flam kann das Programm flam interaktiv oder per Shellscript aufgerufen werden.

Syntax

flam -parameter[=*Wert*] [...]

Da die Angabe der Parameter nicht nur im Kommando, sondern auch über eine Parameterdatei erfolgen kann, und die für die Durchführung einer Funktion dann noch fehlenden Parameter der Defaultwertedatei entnommen werden, ist die Reihenfolge der Angaben beliebig.Die für die

Parameter und Parameterwerte dürfen bei der Eingabe soweit abgekürzt werden, wie die Eindeutikeit ihrer Bedeutung es zulässt. So kann etwa -recs anstelle von -recsize verwendet werden, während bei weniger Zeichen unklar wäre, ob -recdelim, -recformat oder- recsize gemeint sei.

In den folgenden Abschnitten werden die Parameter, die bei der jeweiligen Funktion erlaubt sind, angegeben. Die detallierte Beschreibung aller Parameter in alphabetischer Reihenfolge finden Sie im Abschnitt 2.3.

In diesem Handbuch wurde der besseren Übersicht halber allen Parametern ein Minuszeichen (-) vorangestellt. Seine Verwendung ist optional.

Erläuterung zur Schreibweise:
In den Syntaxbeschreibungen auf auf dieser und den folgenden Seiten sind die in eckigen Klammern gesetzten Syntaxelemente, im jeweiligen Kontext optional. *Kursiv* dargestellte Kommando-Bestandteile sind vom Benutzer durch aktuelle Werte aus dem jeweiligen Wertebereich zu ersetzen. Auslassungszeichen (…) deuten Wiederholungen des Vorhergehenden an.

2.2.1 FLAM Komprimierung

Syntax

FLAM -compress -parameter[=*Wert*] [...]

-attributes = *Attributoption*

-flamcode = *Zeichensatz*

-flamfile = *Dateispezifikation*

-flamin = *Dateispezifikation*

-inrecdelim = *Satztrennzeichen*

-inrecformat = *Satzformat*

-inrecsize = *n*

-kmexit = *Exitangaben*

-maxbuffer = *n*

-maxrecords = *n*

-msgfile = *Meldungsdatei*

-mode = *Komprimierungsmodus*

-ndc

-nopath

-parfile = *Parameterdatei*

-password = *Kennwort*

-recdelim = *Satztrennzeichen*

-recformat = *Satzformat*

-recsize = *n*

-show = *Anzeigeoption*

-translate = *Codetabelle*

2.2.2 FLAM Dekomprimierung

Syntax

FLAM -decompress -parameter[=*Wert*] [...]

-flamfile	= *Dateispezifikation*
-flamout	= *Dateispezifikation*
-kmexit	= *Exitangaben*
-msgfile	= *Meldungsdatei*
-option	= *Flamoption*
-outpath	=*Ausgabeverzeichnis*
-outrecdelim	= *Satztrennzeichen*
-outrecformat	= *Satzformat*
-outrecsize	= *n*
-pad_char	= *Füllzeichen*
-parfile	= *Parameterdatei*
-password	= *Kennwort*
-show	= *Anzeigeoption*
-translate	= *Codetabelle*

2.2.3 Anzeigen der FLAM-Defaultwerten

Syntax

FLAM -list[=*Listspezifikation*]

Siehe Beschreibung des Parameters -list.

2.2.4 Einstellung der FLAM-Defaultwerte

Syntax

FLAM -defaults=*Defaultspezifikation*

Dieser Aufruf darf nur vom Systemmanager benutzt werden. Siehe Beschreibung des Parameters -default.

2.3 Parameter des flam-Kommandos

Im folgenden Teil werden die Parameter von FLAM, unabhängig davon in welcher Funktion sie benutzt werden können, in alphabetischer Reihenfolge beschrieben.

-attributes

Dieser Parameter bewirkt, dass Komprimierungsinformationen in die FLAMFILE eingetragen werden.

Syntax

-attributes=*Attributoption*

Werte

Attributoption	Bedeutung
none	keine Komprimierungsinformationen
common	allgemeine Komprimierungsinformationen
all	allgemeine Komprimierungsinformationen und system-spezifische Informationen über die Originaldatei

Beschreibung

Die Eintragung von Komprimierungsinformationen in die FLAMFILE gestattet eine spätere Extraktion dieser Informationen, auch ohne vollständige Dekomprimierung der FLAMFILE. So können bei der Komprimierung Dateiorganisation, Satzformat und Satzlänge der Originaldatei festgehalten werden sowie ein Kennzeichen, unter welchem Betriebssystem die Komprimierung erfolgt (-attributes= common).

Dadurch kann im Bedarfsfall bei der Dekomprimierung die dekomprimierte Datei mit denselben Charakteristika erzeugt werden.

Optional kann zusätzlich die Dateispezifikation der Originaldatei eingetragen werden (-attributes=all). Dies ermöglicht, bei der Dekomprimierung die Ausgabespezifikation [ ] oder [*] anzugeben, so dass FLAM automatisch auf die eingetragene Dateispezifikation und die zugehörigen Charakteristika zurückgreift.

Mit -attributes=none wird keinerlei Information über die Originaldatei in die FLAMFILE eingetragen.

-compress

Dieser Parameter bewirkt, dass FLAM die Originaldatei komprimiert.

Syntax

-compress

Werte

keine

Beschreibung

Beim Aufruf von FLAM muss -compress angegeben werden, um die Originaldatei zu komprimieren und eine FLAMFILE zu erzeugen.

-decompress

Dieser Parameter bewirkt, dass FLAM die Komprimatsdatei dekomprimiert.

Syntax

-decompress

Werte

keine

Beschreibung

Beim Aufruf von FLAM muss -decompress angegeben werden, um die Komprimatsdatei zu dekomprimieren und eine dekomprimierte Datei zu erzeugen, bzw. die Informationen über die Originaldatei(en) anzuzeigen.

-defaults

Mit diesem Parameter können die installationsspezifischen Standardeinstellungen geändert werden. Dieser Parameter kann nur vom Systemmanager benutzt werden.

Syntax

-defaults=*Defaultspezifikation*

Werte

Defaultspezifikation

Defaultspezifikationen bestehen jeweils aus einem Schlüsselwort oder einem Schlüsselwort mit Wertzuweisung. Die Schlüsselwörter sind bis auf einige Ausnahmen identisch mit den Parametern des FLAM-Kommandos, und die Wertzuweisungen unterliegen derselben Syntax wie bei den Parametern.

Soweit bei nachstehenden Defaultspezifikationen auf die entsprechenden Parameter verwiesen wird, können Syntax und Bedeutung den jeweiligen Beschreibungen entnommen werden.

ascii_ebcdic= ***Übersetzungsoption***

Gibt an, wie einzelne ASCII-Zeichen in EBCDIC-Code umzusetzen sind. Übersetzungsoptionen haben das Format:

ascii_code:ebcdic_code

ascii_code und ebcdic_code sind jeweils der Code des umzusetzenden ASCII-Zeichens bzw. des einzusetzenden EBCDIC-Zeichens, angegeben als Hexadezimalzahl zwischen 00 und ff.

Beispielsweise bewirkt

-defaults=ascii_ebcdic=41:81

dass bei einer Komprimierung mit -translate=e/a jedes ASCII "A" in ein EBCDIC "a" umgesetzt wird.

attributes=*Attributoption*

Siehe Beschreibung des Parameters attributes.

Hinweis:
Es ist zu empfehlen, als Defaultwert -attributes=all einzusetzen, damit FLAMFILES, die unter Benutzung der Defaultwerte erzeugt worden sind und Komprimate mehrerer Originaldateien enthalten, wieder in einzelne Dateien mit den Originalnamen und den Originaldateiattributen, wie z.B. dem Satzformat, dekomprimiert werden können. Andernfalls gehen bei Benutzung der Defaultwerte diese Informationen verloren und die Herkunft der Dateien ist später nicht mehr rekonstruierbar.

compress Siehe Beschreibung des Parameters compress.

decompress Siehe Beschreibung des Parameters decompress.

ebcdic_ascii=*Übersetzungsoption*

Gibt an, wie einzelne EBCDIC-Zeichen in ASCII-Code umzusetzen sind. Übersetzungsoptionen haben das Format:

ebcdic_code:ascii_code

ebcdic_code und ascii_code sind jeweils der Code des umzusetzenden EBCDIC-Zeichens bzw. des einzusetzenden ASCII-Zeichens, angegeben als Hexadezimalzahl zwischen 00 und ff.

flamcode=*Zeichensatz* Siehe Beschreibung des Parameters flamcode.

flamfile=*Dateispezifikation*
flamfile=none

Legt den Standard-Dateinamen für die FLAMFILE fest. Die Dateispezifikation muss eine einzelne Datei eindeutig benennen und darf weder Wildcards enthalten, noch als Substitutionsvorschrift angegeben werden.

Es ist zu beachten, dass bei der Definition eines Standard-Dateinamens für die FLAMFILE stdin bzw. stdout nicht mehr benutzt werden kann.

Mit flamfile=none wird die Verwendung eines Default-wertes für diese Einstellung anulliert.

Dieser Dateiname ist bei der Komprimierung der Defaultwert für die Ausgabedatei und bei der Dekomprimierung der Defaultwert für die Eingabedatei. Die mit recformat, recsize, recdelim, delete, user_io, exd20, und exk20 angegebenen Charakteristika beziehen sich auf diese Datei.

flamin=*Eingabespezifikation*
flamin=none

Legt den Standard-Dateinamen für die Eingabedatei bei der Komprimierung fest. Die Eingabespezifikation ist eine Dateispezifikation. Sie muss eine einzelne Datei eindeutig benennen und keine Wildcards enthalten.

Es ist zu beachten, dass bei der Definition eines Standard-Dateinamens für die Eingabedatei stdin nicht mehr benutzt werden kann.

Mit flamin=none wird die Verwendung eines Defaultwertes für diese Einstellung anulliert.

flamout=*Ausgabespezifikation*
flamout=none

Legt den Standard-Dateinamen für die Ausgabedatei bei der Dekomprimierung fest. Ausgabespezifikation ist eine Dateispezifikation. Sie muss eine einzelne Datei eindeutig benennen und darf keine Wildcards enthalten oder eine Substitutionsvorschrift sein.

Es ist zu beachten, dass bei der Definition eines Stanard-Dateinamens für die Ausgabedatei stdout nicht mehr benutzt werden kann.

Mit flamout=none wird die Verwendung eines Defaultwertes für diese Einstellung anulliert.

inrecdelim
Siehe Beschreibung des Parameters inrecdelim.

inrecformat
Siehe Beschreibung des Parameters inrecformat.

inrecsize
Siehe Beschreibung des Parameters inrecsize.

maxbuffer
Siehe Beschreibung des Parameters maxbuffer.

maxrecords
Siehe Beschreibung des Parameters maxrecords.

mode=*FLAM-Modus*
Siehe Beschreibung des Parameters mode.

msgfile=*Messagefile*
msgfile=none
Siehe Beschreibung des Parameters msgfile.
Mit msgfile=none wird die Verwendung eines Defaultwertes für diese Einstellung anulliert.

option=*FLAM-Option*
Siehe Beschreibung des Parameters option.

outrecdelim
Siehe Beschreibung des Parameters outrecdelim.

outrecformat
Siehe Beschreibung des Parameters outrecformat.

outrecsize
Siehe Beschreibung des Parameters outrecsize.

parfile=*Parameterdatei*
parfile=none
Siehe Beschreibung des Parameters parfile.
Mit parfile=none wird die Verwendung eines Defaultwertes für diese Einstellung anulliert.

recdelim
Siehe Beschreibung des Parameters recdelim.

recformat
Siehe Beschreibung des Parameters recformat.

recsize
Siehe Beschreibung des Parameters recsize.

show=*Anzeigeoption* Siehe Beschreibung des Parameters show.

translate=*Codetabelle*
translate=none Siehe Beschreibung des Parameters translate.
Mit translate=none wird die Verwendung eines Defaultwertes für diese Einstellung anulliert.

Beschreibung

Für alle Parameter des flam-Kommandos, mit Ausnahme der Parameter -defaults, -list, -kmexit, -ndc, -nopath, -outpath und -password können vom Systemmanager Standardeinstellungen (Defaultwerte) festgelegt werden. Die Änderung von Defaultwerten kann nur mittels eines Kommandos erfolgen.

Die Definition von Defaultwerten geschieht durch Aufruf von FLAM mit dem Parameter -defaults unter Angabe der Defaultspezifikation. Diese besteht aus einem Schlüsselwort und evtl. einer Wertzuweisung. Die Syntax für die Defaultspezifikationen stimmt mit derjenigen der FLAM-Parameter überein. Zusätzlich gibt es die Parameter ascii_ebcdic und ebcdic_ascii.

Bei der Definition von Defaultwerten ist zu berücksichtigen, dass manche Einstellungen mittels Defaultwert vom Benutzer weder durch Eingaben in der Kommandozeile noch in der Parameterdatei geändert werden können. Dies ist z. B. bei -defaults=flamin=... der Fall, wo zwar beim Aufruf von FLAM zum Komprimieren oder Dekomprimieren der Dateiname änderbar, aber Eingabe von stdin nicht mehr möglich ist.

-flamcode

Dieser Parameter legt bei der Komprimierung den Zeichensatz für zeichencodierte Informationen, wie Dateinamen oder Steuerzeichen, in der FLAMFILE fest.

Syntax

-flamcode=*Zeichensatz*

Werte

Zeichensatz	**Bedeutung**
ascii	ASCII-Code für die FLAMFILE verwenden.
ebcdic	EBCDIC-Code für die FLAMFILE verwenden.

Beschreibung

Dieser Parameter gibt bei der Komprimierung an, in welchem Zeichensatz zeichencodierte Informationen in der FLAMFILE darzustellen sind. Im adc-, cx8- und vr8-Modus betrifft dies nur Informationen im FLAM-Fileheader, wie etwa den Original-Dateinamen und einige Steuerzeichen, denn das Komprimat wird binär dargestellt. Im cx7-Modus werden auch alle FLAM-Steuerzeichen des Komprimats in diesem Zeichensatz codiert. In allen Modi bleiben hingegen die aus den Originaldaten übernommenen Zeichen unübersetzt. Ihre Übersetzung kann durch den Parameter -translate bewirkt werden.

-flamfile

Mit diesem Parameter wird (werden) die FLAMFILE(s) angegeben.

Syntax

-flamfile=*Dateispezifikation*

Werte

Dateispezifikation

Bei der Komprimierung bezeichnet *Dateispezifikation* eine einzelne Datei oder eine Liste mit mehreren Dateien, durch Kommata getrennt. Auch eine Substitutionsvorschrift ist zulässig (s. Abschnitt 2.4).

Bei der Dekomprimierung bezeichnet die *Dateispezifikation* eine einzelne Datei, ein Suchmuster (mit Wildcards * oder ?) oder eine aus solchen Elementen bestehende Liste, durch Kommata getrennt.

Beschreibung

Bei der Komprimierung (Parameter -compress) werden die komprimierten Daten in die angegebene(n) Datei(en) geschrieben.

Bei der Dekomprimierung (Parameter -decompress) werden die komprimierten Daten aus der (den) angegebenen Datei(en) gelesen.

Enthält die *Dateispezifikation* mehr als eine Datei, sind die in den Kapiteln 2.4.1 bis 2.4.3 angegebenen Regeln zur Syntax der Dateispezifikation und zur Zuordung zur Eingabespezifikation (Komprimierung) bzw. Ausgabespezifikation (Dekomprimierung) zu beachten.

-flamin

Mit diesem Parameter werden die zu komprimierenden Dateien angegeben.

Syntax

-flamin=*Eingabespezifikation*

Werte

Eingabespezifikation

Die Eingabespezifikation bezeichnet eine einzelne Datei, ein Verzeichnis, ein Suchmuster (mit Wildcards * oder ?) oder eine aus solchen Elementen bestehende Liste, durch Kommata getrennt.

Beschreibung

Die angegebenen Dateien werden abhängig von den weiteren Angaben des Kommandos komprimiert. Ist ein Verzeichnis angegeben, werden alle darin und ggf. in dessen Unterverzeichnissen enthaltenen Dateien komprimiert.

Enthält die Eingabespezifikation mehr als ein Element, sind die in Kapitel 2.4.1 und 2.4.3 angegebenen Regeln zur Syntax und Zuordnung zu den Angaben für die FLAMFILE zu beachten.

-flamout

Mit diesem Parameter werden die Namen der dekomprimierten Dateien angegeben.

Syntax

-flamout=*Ausgabespezifikation*

Werte

Ausgabespezifikation

Die Ausgabespezifikation bezeichnet eine einzelne Datei, ein Suchmuster (mit Wildcards * oder ?) oder eine aus solchen Elementen bestehende Liste, durch Kommata getrennt. Sie kann auch aus einer Substitutionsvorschrift oder den speziellen Ausgabespezifikationen [ ], [*] oder [*Dateiname*] bestehen (vgl. Abschnitt 2.4).

Beschreibung

Bei der Dekomprimierung werden in die angegebene(n) Datei(en) die dekomprimierten Daten der FLAMFILE(s) geschrieben.

Enthält die Ausgabespezifikation mehr als eine Datei, sind die in Kapitel 2.4.2 und 2.4.3 angegebenen Regeln zur Syntax und Zuordnung zu den Angaben der FLAMFILE zu beachten.

-inrecdelim

Dieser Parameter gibt ein Satztrennzeichen für Originaldateien an.

Syntax

-inrecdelim=*Satztrennzeichen*

Werte

Satztrennzeichen

Hexadezimale Zeichen zwischen 01 und ff, die Länge kann 1 oder 2 Byte sein.

Beschreibung

Mit diesem Parameter wird das Satztrennzeichen einer Originaldatei mit Satzformat stream angegeben. Bei Satzformat stream ohne Angabe des Satztrennzeichens werden 0x0a und 0x0d0a als Satzende interpretiert.

Soll nur 0x0a als Satzende gelten, muss -inrecdelim=0a angegeben werden, um ein 0x0d vor einem 0x0a nicht als Teil der Satztrennzeichen, sondern als Teil der Daten zu interpretieren. Soll eine Datei nur die Satztrennzeichen 0x0d0a enthalten, muss -inrecdelim=0d0a angegeben werden, um alleinstehende Zeichen 0x0a als Teil der Daten zu erkennen.

Es sind nicht nur 0x0a und 0x0d0a als Satztrennzeichen, sondern auch beliebige andere Zeichenkombinationen erlaubt.

-inrecformat

Mit diesem Parameter wird das Satzformat der Originaldatei angegeben.

Syntax

-inrecformat=*Formatoption*

Werte

Formatoption	**Bedeutung**
eaf	EAF-Daten, variable Satzlänge mit 4 Byte Längenfeld in ASCII- oder EBCDIC-Code. Die Satzlänge gibt nur die Länge der Daten ohne Längenfeld an. Das Längenfeld ist Teil der Daten
fix	feste Satzlänge
stream	variable Satzlänge (maximal 2.048 Zeichen) mit Satztrennzeichen
undefined	Sätze ohne Struktur; es werden Sätze gleicher Länge gelesen, letzter Satz kann kürzer sein
variable	variable Satzlänge mit 2 Byte binärer Satzlänge einschließlich Längenfeld
var_2b	variable Satzlänge mit 2 Byte binärer Satzlänge einschließlich Längenfeld
var2b_data	variable Satzlänge mit 2 Byte binärer Satzlänge **ohne** Längenfeld
var_4b	variable Satzlänge mit 4 Byte binärer Satzlänge einschließlich Längenfeld (Host)
var_ascii	variable Satzlänge mit 4 Byte Längenfeld in ASCII-Code einschließlich Längenfeld
var_ebcdic	variable Satzlänge mit 4 Byte Längenfeld in EBCDIC-Code einschließlich Längenfeld

Beschreibung

Dieser Parameter gibt das Satzformat der Originaldatei an. Mit dem angegebenen Satzformat werden die Daten als logische Sätze von der Datei gelesen.

-inrecsize

Mit diesem Parameter wird die Satzlänge der Originaldatei angegeben.

Syntax

-inrecsize=*n*

Werte

n

Ganze Zahl zwischen 1 und 32760)

Beschreibung

Dieser Parameter gibt die Satzlänge von Originaldateien mit Satzformat fix oder undefined an.

-kmexit

Mit diesem Parameter wird ein Benutzerausgang für automatische Schlüsselverwaltung angegeben.

Syntax

-kmexit=*Exitangaben*

Werte

Exitangaben haben die allemeine Form

[(]*func*[([*lib*])[*exparm*]][)]

Dabei ist

func der Name einer Funktion in der Shared Library, die das zum Ver- oder Entschlüsseln benötigte Kennwort liefert.

lib der Name einer Shared Library, die *func* enthält (ohne Dateityp .so). Bei Fehlen dieser Angabe wird der Standardwert **libflamkm.so** angenommern. FLAM sucht diese Shared Library im Verzeichnis $FLAM_PATH/../lib, wenn die Umgebungsvariable FLAM_PATH definiert ist, sonst in /usr/lib.
Hinweis: Bei Eingabe im Shell-Kommando muss jeder runden Klammer ein Escape-Character (\) vorangestellt werden, damit diese nicht von der Shell interpretiert wird. In einer Parameterdatei sind keine Escape-Character erforderlich.

exparm eine Folge von maximal 256 alphanumerischen Zeichen, die an die Kennwortfunktion als Parameter übergeben wird. Sie darf keine Leerzeichen oder Klammern enthalten. Kommata dürfen verwendet werden, doch müssen dann die gesamten *Exitangaben* in (runden) Klammern eingeschlossen werden. Vor *exparm* muss immer ein Klammerpaar stehen, ggf. auch ohne Inhalt.

Beschreibung

Mit diesem Parameter wird FLAM der Name einer Funktion übergeben, die FLAM dann aufruft, um ein Kennwort für die Ver- oder Entschlüsselung einer Datei zu empfangen. Diese Funktion muss sich in einer Shared Library (.so Datei) befinden, deren Name nach dem Funktionsnamen angegeben werden kann. Ferner können dem Librarynamen Parameter als alphanumerische Zeichenkette nachgestellt werden.

Für -kmexit ist kein Defaultwert zulässig.

Bei Benutzung dieses Parameters darf -password nicht verwendet werden.

Für Details zur Schnittstelle zu diesem Benutzerausgang und dessen Erzeugung siehe Abschnitt 6.2.

-list

Mit diesem Parameter können die installationsspezifischen Defaultwerte angezeigt werden. Die Anzeige von Defaultwerten kann nur mittels eines Kommandos erfolgen.

Syntax

-list[=*Listspezifikation*]

Werte

Listspezifikation

Hierfür kann optional jeder FLAM-Parameter angegeben werden, für den ein Standardwert eingestellt werden kann.

Beschreibung

Die installationsspezifischen Standardeinstellungen können mit dem Parameter list am Bildschirm angezeigt werden. Bei Angabe einer Listspezifikation wird der aktuelle Standardwert angezeigt. Fehlt diese Angabe, wird eine Liste aller aktuellen Standardwerte ausgegeben mit Ausnahme der Codetabellen für ASCII-EBCDIC und EBCDIC-ASCII-Übersetzung.

Ferner wird die FLAM-Version und das Erzeugungsdatum des Programms angezeigt

Bei fehlender Listspezifikation werden die nachstehendem Beispiel entsprechenden Informationen angezeigt, wobei Betriebssystem, Erzeugungsdatum und Standardwerte den aktuellen Gegebenheiten angepasst sind:

```
FLAM (R) 4.2.0 fuer Linux
copyright (c) 2012 by limes datentechnik gmbh
Build from Aug  2 2012 - 14:53:25
Ablaufdatum: 31.12.2012
Lizensiert fuer limes_datentechnik_gmbh

---------------------------------------------
            Standardeinstellungen
---------------------------------------------

attributes ....................... all

decompress

flamcode ......................... ASCII

flamfile (Komprimatsdatei) ....... (not specified)
             recformat ........... fix
             recsize ............. 512

flamin (Originaldatei) ........... (not specified)
             inrecformat ......... stream
             inrecdelim .......... x'0a'

flamout (dekomprimierte Datei) ... (not specified)
             outrecformat ........ stream
             outrecdelim ......... x'0a'

maxbuffer ........................ 32768
maxrecords ....................... 4095

message file ..................... (not specified)

mode ............................. adc
   Verschluesselung .............. keine

option ........................... nocut
                                   nosuppress

parameter file ................... (not specified)

show ............................. all

translate (Konvertierungsdatei) .. (not specified)
```

Mit der Listspezifikation ascii_ebcdic und ebcdic_ascii können die entsprechenden Codetabellen angezeigt werden. Die Anzeige der Codetabellen erfolgt im selben Format, wie es für die Einstellung der entsprechenden Standardwerte beschrieben ist (siehe Beschreibung des Parameters -defaults).

-maxbuffer

Dieser Parameter gibt die Größe des Matrixpuffers an.

Syntax

-maxbuffer=*n*

Werte

n

Ganze Zahl zwischen 1 und 2.621.440. Werte n mit 0≤n≤2.560 werden als Anzahl KBytes (1 KByte = 1.024 Bytes) interpretiert, größere Werte als Anzahl Bytes. FLAM wählt eine der Puffergrößen aus der folgenden Tabelle (in Kbytes):

2	4	6	8	10	12	14	16
32	48	64	80	96	112	128	144
176	224	256	288	320	352	384	416
512	640	768	896	1.024	1.536	2.048	2.560

Ist der angegebene Wert nicht in der Tabelle, so wird, soweit vorhanden, die nächst höhere Puffergröße aus der Tabelle genommen, ansonsten immer das Maximum (2.560).

Beschreibung

Beim Komprimieren mit den Modi cx7, cx8 und vr8 reserviert FLAM den Matrixpuffer für die Zwischenspeicherung der Sätze aus der Originaldatei. Beachten Sie, dass ein Matrixpuffer derselben Größe auch bei der Dekomprimierung verfügbar sein muss, auch wenn diese auf anderen Systemen erfolgt.

Bei Modus adc ist dieser Parameter unwirksam.

-maxrecords

Dieser Parameter gibt die maximale Anzahl der Sätze an, die in einem Komprimatsblock komprimiert werden.

Syntax

-maxrecords=*n*

Werte

n

Ganze Zahl zwischen 1 und 4095.

Beschreibung

FLAM speichert Sätze bis zu der vorgegebenen Anzahl im Matrixpuffer zwischen und komprimiert sie dann. Diese vorgegebene Anzahl kann von Fall zu Fall unterschritten werden, wenn die Matrixpuffergröße für die volle Anzahl von Sätzen nicht ausreicht. Die Komprimierungseffizienz nimmt in der Regel mit der Anzahl zu. Mit -maxrecords=1 wird eine sequentielle, satzweise Komprimierung erreicht.

Bei Modi cx7, cx8 und vr8 werden maximal 255 Sätze je Matrix gespeichert. Wird ein höherer Wert angegeben, wird er nur im Komprimierungsmodus adc wirksam.

-mode

Dieser Parameter gibt den Modus an, in dem die Originaldatei komprimiert und ggf. verschlüsselt wird.

Syntax

-mode=*FLAM-Modus*

Werte

FLAM-Modus	**Bedeutung**
adc	Mit -mode=adc wird die Eingabedatei im adc-Modus komprimiert und eine binäre FLAMFILE erzeugt. Dies ist der effizienteste FLAM-Modus und ist universell einsetzbar, ungeachtet der Datenstruktur. Beide verschlüsselnden Modi (aes, flamenc) basieren auf diesem Kompressionsverfahren.
aes	-mode=aes bewirkt eine Komprimierung im adc-Modus mit AES-Verschlüsselung. Die Angabe eines Kennworts mittels des -password-Parameters oder eines Benutzerausgangs mittels des -kmexit-Parameters ist obligatorisch
flamenc	-mode=flamenc bewirkt eine Komprimierung im adc-Modus mit FLAM-Verschlüsselung (d.i. das in FLAM Version 3 verwendete proprietäre Verschlüsselungsverfahren). Die Eingabe eines Kennworts mittels des -password-Parameters oder eines Benutzerausgangs mittels des -kmexit-Parameters ist obligatorisch
cx7	Mit -mode=cx7 wird die Eingabedatei im cx7-Modus komprimiert und eine zeichencodierte FLAMFILE erzeugt. In diesem Modus sollten jedoch nur Dateien komprimiert werden, die ausschließlich druckbare Zeichen enthalten. Dieser Modus ist etwas weniger effizient, die erzeugte FLAMFILE kann aber ohne Informationsverlust in andere Zeichencodes übersetzt werden, etwa von ASCII nach EBCDIC.
cx8 **vr8**	Mit -mode=cx8 bzw. -mode=vr8 wird die Eingabedatei im cx8- bzw. vr8-Modus komprimiert und eine binäre FLAMFILE erzeugt. Diese beiden Modi dienen zur Wahrung der Kompatiblität mit älteren FLAM-Versionen und sind i.d.R. nicht so effizient wie der adc-Modus.

Beschreibung

Die Eingabedatei wird mit dem angegebenen FLAM-Modus komprimiert und, falls ein Kennwort mittels -password- oder -kmexit-Parameter verfügbar gemacht wird, mit dem gewählten Verfahren verschlüsselt. Wird als Modus adc angegeben, wird bei Eingabe eines Kennworts wie mit -mode=flamenc verschlüsselt.

-msgfile

Mit diesem Parameter kann die Ausgabe von FLAM-Meldungen in eine Datei umgeleitet werden.

Syntax

-msgfile=*Meldungsdatei*

Werte

Meldungsdatei

Spezifikation einer Datei, in die die FLAM-Meldungen auszugeben sind.

Beschreibung

FLAM-Meldungen werden auf stderr ausgegeben, dies ist im Dialogbetrieb in der Regel der Benutzerbildschirm. Mit -msgfile=*Meldungsdatei* kann eine Ausgabe in eine Datei veranlasst werden, die als Permanentkopie zur Dokumentation der Komprimierung dienen kann. Dies ist beispielsweise bei der Stapelverarbeitung von Bedeutung.

Die Protokollierung in der Meldungsdatei beginnt jedoch erst nach Prüfung der Syntax und der Verträglichkeit der im Aufruf benutzten FLAM-Einstellungen. Fehlermeldungen aufgrund von Syntaxfehlern, fehlenden Dateien oder fehlerhaften Einstellungen werden nicht protokolliert. In solchen Fällen bricht FLAM die Verarbeitung ab, ohne eine Meldungsdatei anzulegen.

-ndc

Mit diesem Parameter kann die Komprimierung im adc-Modus unterdrückt werden.

Syntax

-ndc

Werte

keine

Beschreibung

Bei Daten, deren geringe Komprimierbarkeit von vornherein feststeht ist, kann die Verarbeitung durch mit diesem Parameter beschleunigt werden, der die Kompression unterbindet. In Verbindung mit den Modi aes und flamenc werden die Daten lediglich verschlüsselt; mit mode=adc bewirkt dieser Parameter, dass FLAM die Daten lediglich kopiert.

Bei allen anderen Modi hat -ndc keinerlei Wirkung.

Mit -ndc erzeugte Komprimate sind in jedem Fall Secure FLAMFILEs und werden bei der Dekomprimierung als adc-Komprimate ausgewiesen.

-ndc kann nicht als Standardwert eingestellt werden.

-nopath

Dieser Parameter bewirkt, dass die Komprimatsdatei den Namen der Originaldatei ohne Pfad enthält.

Syntax

-nopath

Werte

keine

Beschreibung

In der Regel trägt FLAM beim Koprimieren den Namen der Originaldatei einschließlich des vollständigen Pfads in den Fileheader ein. Beim Dekomprimieren auf Systemen mit anderer Verzeichnisstruktur oder anderen Syntaxregeln für Dateinamen kann es dadurch zu Problemen kommen, wenn versucht wird, den kompletten Namen zu übernehmen. Mit –nopath wird die Pfadinformation des Originalnamen ausgeblendet und nur der reine Dateiname eingetragen.

Beim Dekomprimieren von mit –nopath komprimierten Dateien, hat –flamout=[*] dieselbe Wirkung wie –flamout=[ ].

-nopath kann nicht als Standardwert eingestellt werden.

-option

Dieser Parameter steuert das Verhalten von FLAM bei der Ausgabe dekomprimierter Sätze.

Syntax

-option=*FLAM-Option*

Werte

Flam-Option	**Bedeutung**
cut	Sätze mit einer Satzlänge, die größer ist als die maximale Satzlänge, werden verkürzt.
nocut	Sätze mit einer Satzlänge, die größer ist als die maximale Satzlänge, werden nicht verkürzt.
suppress	Leerzeichen am Satzende werden unterdrückt.
nosuppress	Leerzeichen am Satzende werden nicht unterdrückt.

Beschreibung

Bei der Dekomprimierung können Satzformat und Satzlänge der dekomprimierten Datei explizit so vorgegeben oder von Vorversionen übernommen werden, dass die maximale Satzlänge kürzer ist als in der Originaldatei. Mit der Option cut wird die Dekomprimierung fortgesetzt, wobei überlange Sätze auf die maximale Länge abgeschnitten werden. Mit nocut wird die Dekomprimierung mit einem Fehlerhinweis abgebrochen.

Wird bei der Dekomprimierung die dekomprimierte Datei mit dem Satzformat stream geschrieben, können mit suppress Leerzeichen am Satzende unterdrückt werden. Mit nosuppress werden sie geschrieben.

-outpath

Dieser Parameter gibt ein Verzeichnis vor, in dem die dekomprimierten Dateien erzeugt werden sollen.

Syntax

-outpath=*Ausgabeverzeichnis*

Werte

Ausgabeverzeichnis

Spezifikation eines Verzeichnispfads, in den die dekomprimierte(n) Datei(en) geschrieben werden soll(en). Dies kann ein absoluter oder ein relativer Pfad sein.

Beschreibung

Wird mit dem Parameter –flamout=... ein Dateiname oder eine Liste angegeben, können diese Angaben auch einen Verzeichnispfad mit vorgeben. Bei –flamout=[ ] ist eine solche Vorgabe nicht möglich. In diesem Fall ist mit Hilfe des Parameters –outpath=... eine solche Vorgabe möglich.

Für -outpath kann kein Standardwert eingestellt werden.

-outrecdelim

Dieser Parameter gibt ein Satztrennzeichen für dekomprimierte Dateien an.

Syntax

-outrecdelim=*Satztrennzeichen*

Werte

Satztrennzeichen

Hexadezimale Zeichen zwischen 01 und ff, die Länge kann 1 oder 2 Byte sein

Beschreibung

Mit diesem Parameter wird das Satztrennzeichen der dekomprimierten Datei mit Satzformat stream angegeben.

Bei Satzformat stream ohne Angabe des Satztrennzeichens wird 0x0a als Satztrennzeichen geschrieben. Mit der Angabe -outrecdelim=0d0a kann die dekomprimierte Datei in MS-DOS-Format erstellt werden.

Daneben ist jede andere Zeichenkombination als Satztrennzeichen erlaubt.

-outrecformat

Mit diesem Parameter wird das Satzformat der dekomprimierten Datei angegeben.

Syntax

-outrecformat=*Formatoption*

Werte

Formatoption

Siehe Beschreibung von *Formatoption* für –inrecformat.

Beschreibung

Dieser Parameter gibt das Satzformat der dekomprimierten Datei an. Mit dem angegebenen Satzformat werden die dekomprimierten Daten als logische Sätze in die Datei geschrieben.

-outrecsize

Mit diesem Parameter wird die Satzlänge der dekomprimierten Datei angegeben.

Syntax

-outrecsize=*n*

Werte

n

Ganze Zahl zwischen 1 und 32760

Beschreibung

Dieser Parameter gibt die Satzlänge der dekomprimierten Dateien mit Satzformat fix oder undefined an. Bei anderen Satzformaten gibt er die Pufferlänge vor.

-pad_char

Mit diesem Parameter wird das Füllzeichen zum Verlängern von Sätzen der dekomprimierten Datei angegeben.

Syntax

-pad_char=*Hexcode*

Werte

Hexcode

Zwei Hexadezimale Ziffern zwischen 00 und ff

Beschreibung

Bei der Dekomprimierung können Satzformat und Satzlänge der dekomprimierten Datei explizit so angegeben werden, dass die Satzlänge größer ist als die der Originaldatei. In diesem Fall werden die Sätze bis zur vorgegebenen Länge mit dem Füllzeichen aufgefüllt.

-parfile

Mit diesem Parameter können die Angaben der Kommandozeile durch FLAM-Parameter in einer Parameterdatei ergänzt oder überschrieben werden.

Syntax

-parfile=*Parameterdatei*

Werte

Parameterdatei

Spezifikation einer Datei, die FLAM-Parameter enthält. Wird kein Pfad angegeben, wird diese im aktuellen Verzeichnis gesucht.

Beschreibung

Durch Angabe einer Parameterdatei können Defaultwerte ohne umfangreiche Eingaben in der Kommandozeile überlagert werden (Siehe hierzu im Abschnitt 2.5, Prioritätsregeln für FLAM-Einstellungen). Diese kann mit einem Texteditor interaktiv erstellt werden (s. Beispiel in Abschnitt 2.6).

Eine Parameterdatei kann nicht auf eine andere Parameterdatei verweisen, darf also keine parfile-Spezifikation enthalten. Ferner darf sie den Parameter –show nicht verändern, das dessen Einstellung schon nach der Analyse der Kommandozeile feststehen muss.

-password

Mit diesem Parameter kann bei Komprimierung und Dekomprimierung mit Modus adc ein Kennwort zum Ver- und Entschlüsseln der FLAMFILE mit dem AES- oder dem FLAMenc-Verfahren angegeben werden.

Syntax

-password=*Kennwort*

Werte

Kennwort

Das Kennwort kann in zweierlei Formaten angegeben werden:

- als einfache Zeichenfolge

Bei diesem Format wird das Kennwort unmittelbar hinter das Gleichheitszeichen gesetzt und kann aus höchstens 64 Zeichen bestehen. Gültige Zeichen sind hierbei lateinische Groß- und Kleinbuchstaben, Dezimalziffern, Bindestrich (-) und Unterstrich (_). Die Verwendung anderer Zeichen kann unerwünschte Effekte zur Folge haben. Groß- und Kleinbuchstaben gelten als verschieden.

Beispiel: -password=Alligator

- als Folge von Hexadzimalziffern

Bei diesem Format kann das Kennwort aus einer geraden Anzahl von bis zu 128 Hexadezimalziffern bestehen (0-9, a-f, A-F). In diesem Kontext gelten Groß- und Kleinbuchstaben als äquivalent. Die Folge wird in Hochkommata (‘) eingeschlossen, und ihr wird ein “X“ vorangestellt. Jede Kombination der gültigen Zeichen ist hierbei zulässig.

Hinweis: Bei Eingabe im Shell-Kommando muss jedem Hochkomma ein Escape-Character (\) vorangestellt werden, damit dieses nicht von der Shell interpretiert wird. In einer Parameterdatei sind keine Escape-Character erforderlich.

Beispiel: password=X\‘416c6c696761746f72\‘

Beschreibung

FLAM benutzt das Kennwort während der Komprimierung mit Modus adc zum Verschlüsseln und während der Dekomprimierung zum Entschlüsseln der Daten. Das beim Dekomprimieren angegebene Kennwort muss mit dem der Komprimierung identisch sein, sonst wird eine entsprechende Fehlermeldung ausgegeben

Mit anderen Modi als adc, aes oder flamenc führt dieser Parameter zu einer Fehlermeldung. Bei Angabe eines Kennworts mit mode=adc wird mit dem flamenc-Verfahren verschlüsselt.

Ein als einfache Zeichenfolge eingebenes Kennwort gilt als identisch mit dem Kennwort, das aus den Hexadezimalziffern besteht, die den Zeichencodes der einzelnen Zeichen entsprechen.

Es ist zu beachten, dass eine Zeichenfolge, die auf einem ASCII-System eingegeben wird, nicht identisch ist mit der gleichen Zeichenfolge auf einem System, das EBCDIC-Code verwendet. Daher sollte für den Datenaustausch zwischen Systemen mit unterschiedlichen Zeichencodes die hexadezimale Schreibweise benutzt werden.

Für password ist kein Defaultwert zulässig.

Bei Benutzung dieses Parameters darf -kmexit nicht verwendet werden.

-recdelim

Dieser Parameter gibt das Satztrennzeichen der FLAMFILE an.

Syntax

-recdelim=*Satztrennzeichen*

Werte

Satztrennzeichen

Zwei oder vier hexadezimale Ziffern zwischen 01 und ffff

Beschreibung

Mit diesem Parameter kann beim Satzformat stream das Satztrennzeichen angegeben werden.

Wird kein Satztrennzeichen beim Satzformat stream angegeben, wird 0x0a verwendet.

Z.B. kann durch Angabe von -recdelim=0d0a die Ausgabe der Daten im MS-DOS-Format erfolgen.

Dieser Parameter kann nur bei der Komprimierung im Modus cx7 angegeben werden.

-recformat

Mit diesem Parameter wird das Satzformat der FLAMFILE angegeben.

Syntax

-recformat=*Formatoption*

Werte

Formatoption	**Bedeutung**
fix	Sätze mit fester Satzlänge
variable	Sätze mit variabler Satzlänge, das Satzlängenfeld ist 2 Byte lang
stream	Sätze mit variabler Satzlänge und Satztrennzeichen (nur mit -mode=cx7 zulässig)

Beschreibung

Die FLAMFILE enthält immer Sätze gleicher Länge.

Beim Modus cx8 und vr8 können die Sätze mit (variable) oder ohne (fix) Satzlängenfeld geschrieben werden.

Beim Modus cx7 wird das Satzformat stream verwendet. Wenn keine Angabe von recdelim erfolgt, wird als Satztrennzeichen 0x0a benutzt.

Die Angabe -recformat ist nur bei der Komprimierung erforderlich.

-recsize

Mit diesem Parameter wird die Satzlänge der FLAMFILE angegeben.

Syntax

-recsize=*n*

Werte

n

Ganze Zahl zwischen 80 und 4095 bei Modus cx7

Ganze Zahl zwischen 80 und 32760 sonst

Beschreibung

Dieser Parameter gibt die Satzlänge der FLAMFILE an.

Unabhängig vom Satzformat werden die komprimierten Daten als Sätze mit gleicher Länge geschrieben.

Eine Beziehung zwischen Komprimatsblöcken und Sätzen in der komprimierten Datei besteht nicht. Ein Satz kann Daten aus einem oder mehreren Komprimatsblöcken enthalten. Ein Komprimatsblock kann in einem oder mehreren Sätzen stehen.

Es besteht keine Beziehung zwischen den Sätzen der komprimierten und unkomprimierten Dateien.

Dieser Parameter ist nur bei der Komprimierung erforderlich.

-show

Dieser Parameter gibt an, welche Informationen von FLAM angezeigt werden sollen.

Syntax

-show=*Anzeigeoption*

Werte

Anzeigeoption	Bedeutung
none	keine Anzeige
all	Zeigt alle verfügbaren Informationen an. Diese hängen von der aufgerufenen Operation ab (Komprimierung oder Dekomprimierung), nicht aber davon, ob FLAM direkt oder mit Parameterdatei aufgerufen wurde.

Anzeige bei der Komprimierung:

FLAM-Version

FLAM-Einstellungen

- Operation (Komprimierung)
- Zeichencode
- ggf. Codetabelle
- Komprimierungsmodus
- ggf. Verschlüsselungsverfahren
- Matrixpuffergröße
- Max. Anzahl Sätze/Matrixpuffer
- Zeichensatz (ASCII oder EBCDIC)
- Verwendete Einstellung von attributes

Name der Originaldatei

Formatangaben für die Originaldatei

Name der Komprimatsdatei (FLAMFILE)

Formatangaben der Komprimatsdatei

Statistik-Informationen (siehe show=statistics)

Fehler- und Warnhinweise

Anzeige-Option	Bedeutung
	Anzeige bei der Dekomprimierung: FLAM-Version FLAM-Einstellungen: Operation (Dekomprimierung) Komprimierungsmodus ggf. Verschlüsselungsverfahren Zeichencode ggf. Codetabelle Name der Komprimatsdatei (FLAMFILE) Name der dekomprimierten Datei ursprünglicher Dateiname Formatangaben für die dekomprimierte Datei Gespeicherte Komprimierungsinformationen (siehe show=attributes) Statistik-Informationen (siehe show=statistics) Fehler- und Warnhinweise
attributes	(nur bei Dekomprimierung) Unterdrückt die Erzeugung der dekomprimierten Datei Zeigt nur gespeicherte Komprimierungsinformationen an Name der Originaldatei (nur wenn mit attributes=all komprimiert wurde) Formatangaben der Originaldatei Komprimierungsmodus ggf. Verschlüsselungsverfahren Zeichensatz (ASCII oder EBCDIC) System, das die FLAMFILE erzeugt hat
error	Zeigt nur Fehler- und Warnhinweise.

statistics

Zeigt Fehler- und Warnhinweise und folgende Statistik an:

Anzahl komprimierter Sätze

Anzahl komprimierter Bytes

Anzahl unkomprimierter Sätze

Anzahl unkomprimierter Bytes

Komprimierungseffizienz im Vergleich zur Originaldatei (nur bei Komprimierung) [1]

benötigte Zeit

[1] Diese Effizienz wird als Verhältnis der effektiv gelesenen, zu der ausgegebenen Byteanzahl in Prozent angegeben. Satzlängenfelder bzw. Satztrennzeichen sowie Zeichen zum Auffüllen von Komprimatssätzen werden nicht berücksichtigt.

Beschreibung

Mit diesem Parameter steuern Sie die Informationsausgabe von FLAM. Sie können wählen zwischen umfassender Information (all), Fehler- und Warnhinweisen mit oder ohne Statistik (error bzw. statistics) und keinerlei Ausgabe (none).

Bei der Dekomprimierung haben Sie die Möglichkeit, sich lediglich Informationen über die Originaldatei anzeigen zu lassen, ohne eine dekomprimierte Datei zu erzeugen. Dazu verwenden Sie -show=attributes. Enthält die FLAMFILE Komprimate mehrerer Dateien, so werden die Informationen über alle enthaltenen Originaldateien nacheinander angezeigt. Eine Anzeige dieser Informationen erfolgt jedoch nur, wenn diese bei der Komprimierung in die FLAMFILE mit eingetragen wurden (siehe Parameter attributes). Bei Angabe von -show=attributes wird der Parameter Ausgabedatei ignoriert.

Die Ausgabe der Meldungen kann durch die Angabe 2 >> Dateiname auf eine Datei umgelenkt werden. Hingegen werden bei Angabe von -msgfile=Dateiname die FLAM-Meldungen in eine Datei geschrieben, während Systemmeldungen weiterhin am Bildschirm erscheinen.

-translate

Dieser Parameter bewirkt eine Codeübersetzung der Daten der Originaldatei vor der Komprimierung oder der Daten der dekomprimierten Datei nach ihrer Dekomprimierung.

Syntax

-translate=*Codetabelle*

Werte

Codetabelle

Spezifikation einer Datei, die die Codetabelle enthält. Die Codetabelle ist eine Zeichenkette von 256 Zeichen des Zeichensatzes, in den die Daten übersetzt werden sollen. (Näheres über die Codetabelle siehe im Anhang B, Codetabellen.)

Beschreibung

Wird -translate=Codetabelle bei der Komprimierung angegeben, so wird jeder Satz der Originaldatei vor der Zwischenspeicherung in den Matrixpuffer gemäß der spezifizierten Codetabelle übersetzt. Bei der Dekomprimierung bewirkt diese Angabe, dass jeder dekomprimierte Satz entsprechend der Codetabelle übersetzt wird, ehe er in die dekomprimierte Datei eingefügt wird.

Standardmäßig werden mit der Installation von FLAM Codetabellen für die Übersetzung von ASCII nach EBCDIC und für die Übersetzung von EBCDIC nach ASCII zur Verfügung gestellt.

Soll eine der in der Defaultwerte-Datei vorhandenen Codetabellen benutzt werden, wird

a/e für die Konvertierung von ASCII nach EBCDIC,
e/a für die Konvertierung von EBCDIC nach ASCII
angegeben.

2.4 Dateispezifikationen

Bei den Dateispezifikationen werden Eingabe- und Ausgabespezifikationen unterschieden.

Bei der Komprimierung wird die Eingabespezifikation (Originaldatei) mit -flamin, die Ausgabespezifikation (Komprimatsdatei) mit -flamfile angegeben.

Bei der Dekomprimierung wird die Eingabespezifikation (Komprimatsdatei) mit -flamfile, die Ausgabespezifikation (dekomprimierte Datei) mit -flamout angegeben.

2.4.1 Eingabespezifikation

Die Eingabespezifikation kann aus einem Datei- oder Verzeichnisnamen, einem Suchmuster (mit Wildcards) oder einer Liste dieser Elemente bestehen. In einer Liste werden die einzelnen Elemente durch Kommata getrennt.

Dateinamen dürfen auch Leerzeichen enthalten, nicht jedoch als erstes Zeichen. Dateinamen, die Leerzeichen enthalten, müssen in Anführungszeichen eingeschlossen sein.

Als Eingabedateien müssen spezifizierte Dateien existieren. Zu Mustern muss mindestens eine passende Datei existieren. Andernfalls bricht die Ausführung des flam-Kommandos unter Ausgabe einer entsprechenden Fehlermeldung ab.

Werden mit der Eingabespezifikation Dateien für die Verarbeitung durch FLAM vorgegeben, muss der Parameter -flamin bei der Komprimierung bzw. -flamfile bei der Dekomprimierung angegeben werden.

Soll die Eingabe der Daten über die spezielle Datei stdin erfolgen, entfällt der Parameter für die Eingabespezifikation. Somit kann FLAM als Filter benutzt werden.

2.4.2 Ausgabespezifikationen

Die Ausgabespezifikation kann aus einem Dateinamen, einer Liste von Dateinamen (durch Kommata getrennt) oder einer Substitutionsvorschrift bestehen. Eine Substitutionsvorschrift hat folgendes Format:

[Zeichenkette1=Zeichenkette2]

Ferner kann eine der (nachstehend beschriebenen) Ausgabespezifikationen [ ], [*] oder [*Dateiname*] angegeben werden.

Ausgabedateien werden neu angelegt, falls sie noch nicht existieren. Im anderen Fall werden sie überschrieben.

Eine Substitutionsvorschrift als Ausgabespezifikation bedeutet, dass der Dateiname der FLAMFILE bzw. der dekomprimierten Datei zu bilden ist durch Ersetzung der Zeichenkette1 durch die Zeichenkette2 im Namen der Eingabedatei. Die Zeichenketten und das Gleichheitszeichen müssen ohne Leerstellen unmittelbar aufeinanderfolgen. Die Substitution wird nur angewandt auf den Dateinamen, aber nicht auf Pfadnamen.

Beispielsweise werden mit dem Kommando

```
flam –compress –mode=cx8
–flamin=helptxt1.txt,helptxt2.txt
–flamfile=[.txt=.cmp]
```

die Dateien helptxt1.txt und helptxt2.txt komprimiert und die Komprimate in den zugehörigen FLAMFILEs helptxt1.cmp und helptxt2.cmp abgelegt.

Die Substitution muss auf alle zugeordneten Eingabedateien anwendbar sein, d.h. Zeichenkette1 muss in den Namen aller zugeordneten Eingabedateien enthalten sein, andernfalls bricht die Ausführung des flam-Kommandos für diese Datei unter Ausgabe einer entsprechenden Fehlermeldung ab. Bei der Bildung des Namens der Ausgabedatei, wird die Substitution nur einmal ausgeführt, auch wenn Zeichenkette1 im Namen der Eingabedatei mehrfach vorkommt.

Spezielle Ausgabespezifikationen sind [*Dateiname*], [ ] und [*].

Mit [*Dateiname*] als Ausgabespezifikation beim Dekomprimieren kann aus einer FLAMFILE, die Komprimate mehrerer Originaldateien enthält, eine einzelne oder - mittels Wildcards - ggf. mehrere Dateien selektiv extrahiert werden. Diese werden im aktuellen Verzeichnis erzeugt, falls keine andere Vorgabe mittels –outpath=... besteht.

Bei Angabe der Ausgabespezifikation [*] beim Dekomprimieren werden die Namen der dekomprimierten

Dateien einschließlich Pfad der FLAMFILE entnommen, sofern diese sie enthält. Andernfalls wird eine Fehlermeldung ausgegeben. Um die Namen der Originaldateien in die FLAMFILE aufzunehmen, muss bei der Komprimierung -attributes=all angegeben werden.

Die Ausgabespezifikation [ ] wirkt ähnlich wie [*], d.h. die Namen für die dekomprimierten Dateien werden aus der bzw. den FLAMFILE(s) entnommen, mit dem Unterschied, dass lediglich der Dateiname berücksichtigt wird. Der Pfadname wird ignoriert. Somit werden alle Dateien im aktuellen Verzeichnis erzeugt, wenn nicht mittels –outpath ein bestimmtes Ausgabeverzeichnis vorgegeben wird.

Voraussetzung für die Verwendung dieser Ausgabespezifikationen ist, dass mit -attributes=all komprimiert wurde.

Die Ausgabespezifikationen [*Dateiname*], [ ] und [*] dürfen nicht Teil einer Liste sein, d.h. sie müssen alleine stehen.

Werden mit der Ausgabespezifikation Dateien für die Verarbeitung durch FLAM vorgegeben, muss bei der Komprimierung der Parameter -flamfile bzw. bei der Dekomprimierung der Parameter -flamout angegeben werden.

Soll die Ausgabe der Daten auf die spezielle Datei stdout erfolgen, entfällt der Parameter für die Ausgabespezifikation. Somit kann FLAM als Filter benutzt werden.

2.4.3 Zuordnung von Eingabe- zu Ausgabespezifikationen

Enthält das flam-Kommando mehrere Ein- und Ausgabespezifikationen, so wird zwischen diesen eine Zuordnung getroffen. Diese legt fest, wo die Komprimate der einzelnen Originaldateien bzw. wo die dekomprimierten Daten der Komprimate gespeichert werden.

Die Zuordnung von Ein- und Ausgabespezifikation zueinander erfolgt über deren Position in der jeweiligen Liste. Die erste Eingabespezifikation wird der ersten Ausgabespezifikation zugeordnet, die zweite Eingabespezifikation der zweiten Ausgabespezifikation usw. Wurden mehr Ein- als Ausgabespezifikationen angegeben, so werden die überzähligen Eingabespezifikation der letzten Ausgabespezifikation zugeordnet. Überzählige Ausgabespezifikationen werden ignoriert.

Ist die Ausgabespezifikation eine Dateispezifikation, so werden sämtliche Ausgaben aus der Verarbeitung der ihr zugeordneten Eingabedateien in dieser Datei abgelegt.

Hierbei ist auf die Verträglichkeit der Dateiattribute zu achten.

Wird als Ausgabespezifikation eine Substitutionsvorschrift angegeben, so wird bei der Komprimierung zu jeder der zugeordneten Eingabedateien eine separate FLAMFILE erzeugt, die entsprechend der Substitutionsvorschrift benannt wird.
Bei der Dekomprimierung wird die Substitutionsvorschrift auf die Namen der FLAMFILEs, nicht auf die Namen der komprimierten Originaldateien angewendet. D.h. aus jeder FLAMFILE, die dem Auswahlmuster entspricht, resultiert eine dekomprimierte Datei.

Ist einer FLAMFILE, die Komprimate mehrerer Dateien enthält, eine Ausgabedatei zugeordnet, so werden sämtliche in der FLAMFILE enthaltenen Komprimate in diese Ausgabedatei dekomprimiert. Sollen wieder einzelne Dateien erzeugt werden, muss eine der Ausgabespezifikationen [] oder [*] vorgegeben werden. In diesem Fall wird jedes Komprimat in eine Datei mit dem ursprünglichen Namen dekomprimiert.

Im nachstehenden Beispiel enthält das flam-Kommando jeweils eine Liste von Eingabe- und Ausgabespezifikationen.

```
flam -compress -attributes=all

-flamin=*.dat,*.txt,func1.hlp,func2.hlp

-flamfile=[.dat=.cmp],text.arc,[hlp=xyz]
```

Die Liste der Eingabespezifikationen enthält vier Angaben, die der Ausgabespezifikationen, zwei Substitutionsvorschriften und eine Dateispezifikation. Entsprechend den Zuordnungsregeln wird für jede Datei, deren Name mit der Zeichenfolge .dat endet, eine gleichnamige Datei, deren Name mit der Zeichenfolge .cmp endet, erzeugt, während die Komprimate aller Dateien, deren Namen mit der Zeichenfolge .txt enden, in einer einzigen FLAMFILE, nämlich in text.arc gespeichert werden. Das Komprimat von func1.hlp wird aufgrund der Substitutionsvorschrift in der letzten Ausgabespezifikation in func1.xyz abgelegt. Da keine weiteren Ausgabespezifikationen folgen, wird diese Vorschrift auch auf func2.hlp angewandt, d.h. für deren Komprimat wird die FLAMFILE func2.xyz erzeugt.

Es ist zu beachten, dass eine Dekomprimierung von text.arc in einzelne Dateien nur möglich ist, wenn bei der Komprimierung attributes=all angegeben wird und damit die Namen aller Originaldateien in die FLAMFILE übernommen werden. Bei der Dekomprimierung darf eine der Ausgabespezifikationen [] oder [*] angegeben werden, damit die Dateien unter Verwendung ihres ursprünglichen Namens einzeln dekomprimiert werden. Würde eine Ausgabedatei spezifiziert, entstünde eine einzige Datei, in der alle Sätze der Ursprungsdateien aneinandergefügt sind.

2.5 Prioritätsregeln für FLAM-Einstellungen

Beim flam-Kommando können Einstellungen, wie die Wahl der Operation (Komprimierung oder Dekomprimierung), Ein- und Ausgabedateien etc., direkt im Kommando oder über eine Parameterdatei explizit vorge- nommen werden.

Ferner sind durch Weglassen bestimmter Angaben implizite Einstellungen möglich, wie etwa der Komprimierungsmodus bei der Komprimierung.

Die effektiven Einstellungen bei fehlenden oder redundanten Angaben werden durch Auswertung verschiedener Informationen in einer festgelegten Reihenfolge ermittelt, aus der eine Prioritätsrangfolge resultiert, die sich auf die Kategorie wie auch auf den Ursprung der Informationen bezieht.

Da Parameterdateien auf andere Parameterdateien verweisen können, ist eine Verkettung mehrerer Parameterdateien möglich, die u.U. widersprüchliche Angaben enthalten können. Hier hat die erste Parameterdatei vor allen anderen Vorrang. Bei mehrfacher Angabe derselben Parameter innerhalb einer Parameterdatei wird jedoch nur die letzte Angabe wirksam.

Für die Bestimmung der FLAM-Einstellungen werden die Informationen in folgender Rangfolge ausgewertet:

1. **die Kommandozeile,**

2. **Parameterdateien,**

3. **installierte Defaultwerte**

Soweit widersprüchliche oder unverträgliche Angaben im Kommando selbst erkennbar sind, wird das Kommando mit entsprechender Fehlermeldung abgewiesen.

In allen übrigen Situationen gelten folgende Prioritätsregeln:

1. Eine im Kommando angegebene FLAM-Operation (Komprimierung/Dekomprimierung) hat Vorrang vor allen anderen Angaben. Fehlt im Kommando diese Angabe, wird sie ggf. in der durch –parfile=... oder als Defaultwert angegebene Parameterdatei gesucht.

   Ist die Operation auch nicht als FLAM-Parameter angegeben, wird für sie der installierte Defaultwert, d.h. die Standardoperation eingesetzt.

2. Angaben, die mit der auszuführenden FLAM-Operation unverträglich sind, werden abgewiesen.

3. Fehlende Angaben im Kommando werden nach Maßgabe der Parameterdatei eingestellt, falls eine solche im Kommando oder als installationsspezifischer Defaultwert angegeben ist.

4. Alle nach Anwendung der Regeln 1-3 noch nicht oder nicht vollständig bestimmten Einstellungen werden entsprechend den installationsspezifischen Defaultwerten gesetzt.

5. Bei fehlender Ein- bzw. Ausgabespezifikation werden die speziellen Dateien stdin bzw. stdout benutzt.

Bei den FLAM-Parametern werden Angaben niedrigerer Priorität stets vollständig durch die höherer Priorität überlagert, d.h. es ist nicht möglich, etwa eine Liste von Eingabespezifikationen teils im Kommando, teils mittels Defaultwerten oder Angaben in der Parameterdatei anzugeben. Die Liste im Kommando annulliert diejenige der niedrigeren Spezifikationsebene.

2.6 Die FLAM-Parameterdatei

Die Parameterdatei wird mit einem Texteditor bearbeitet und enthält FLAM-Parameter, die eindeutig den Angaben in der Kommandozeile des flam-Kommandos entsprechen.

Jeder FLAM-Parameter besteht aus einem Schlüsselwort, gegebenenfalls mit einer Wertzuweisung oder einer Werteliste. Die FLAM-Parameter können so angeordnet werden, dass jede Zeile einen FLAM-Parameter enthält; eine Zeile kann aber auch mehrere durch Kommata getrennte Parameter enthalten. Kommentare sind erlaubt und beginnen mit einem "!". Ende eines Kommentars ist das Zeilenende.

Die Länge einer Zeile der Parameterdatei darf 2.048 Bytes nicht übersteigen.

Das Beispiel illustriert die Verwendung einer Parameterdatei. FLAM wird per Kommando aufgerufen:

```
FLAM –parfile=param.FLAM
```

Die Parameterdatei param.flam enthält folgende FLAM-Parameter:

```
compress
mode=cx8
maxbuffer=128
flamin=test.dat
flamfile=flamfile1.cmp
attributes=all
recformat=fixed
recsize=1024
```

Dieser Aufruf von FLAM bewirkt die Komprimierung der Datei test.dat im Modus cx8. Die FLAMFILE ist die Datei flamfile1.cmp. Sie hat ein fixes Satzformat und eine Satzlänge von 1.024 und enthält alle Komprimierungsinformationen, so dass in derselben Umgebung eine Dekomprimierung ohne Spezifikation der dekomprimierten Datei möglich ist.

Dieselben FLAM-Parameter können auch in folgender Form angeordnet werden:

compress,mode=cx8,maxbuffer=128

flamin=test.dat

flamfile=flamfile1.cmp

recformat=fix,recsize=1024

attributes=all

Auf dem Bildschirm würde folgende Ausgabe erscheinen:

```
FLAM (R) 4.2.0 fuer Linux
copyright (c) 2012 by limes datentechnik gmbh
Start: 06.08.2012  15:35:07

--------
compress
--------
mode ........................... adc
maxbuffer ...................... 65,536
maxrecords ..................... 4,096
flamcode ....................... ASCII
attributes ..................... all

Name Originaldatei:.............  /path001/test.dat
             inrecformat ........ stream
             inrecdelim ......... x'0a'

Anzahl unkomprimierte Saetze ..... 778
Anzahl unkomprimierte Bytes ...... 26,947

Name Komprimatsdatei: ........... flamfile1.cmp
             recformat .......... fix
             recsize ............ 1,024

Anzahl komprimierte Saetze ....... 14
Anzahl komprimierte Bytes ........ 7,168

Effizienz in Prozent ............ 73,4
Laufzeit in Sek. ................ 0.0
```

2.7 Die FLAM-Defaultwertedatei

Defaultwerte für das flam-Kommando werden bei der Installation von FLAM in der Defaultwertedatei flam_def.dat gespeichert, die im Unterverzeichnis lib des Installationspfads angelegt wird.

Diese Datei wird mit den Zugriffsrechten -rw-r--r-- angelegt. Sie kann jederzeit vom Systemmanager geändert, darf aber nicht gelöscht werden.

Zur Änderung der Defaultwertedatei wird das flam-Kommando mit dem Parameter -defaults verwendet.

Die Einstellungen in der Defaultwertedatei können durch das Kommando flam -list angezeigt werden. Hierfür sind keine besonderen Privilegien erforderlich.

Beim Aufruf von FLAM werden Angaben, für die weder aus der Kommandozeile noch in einer Parameterdatei Einstellungen vorliegen, erforderlichenfalls aus der Defaultwerte-Datei flam_def.dat entnommen, die die installationsspezifischen Standardeinstellungen enthält.

Diese Datei ist binär und darf nicht mit anderen Programmen als flam geändert werden.

2.8 Alternative Bezeichnungen von Parametern

Zum Teil werden alte Parameter durch neue ersetzt, um die erweiterte Funktionalität von FLAM besser benutzen zu können, z.T. sind auch systemspezifisch für die gleiche Funktion andere Bezeichnungen gewählt worden. Um zur Version 2.0 von FLAM auf UNIX und auch zu anderen Systemen kompatibel zu sein, sind nicht nur die genannten Parameterbezeichnungen zulässig, sondern auch die alten und die anderer Systeme. Zu empfehlen ist allerdings die Benutzung der bereits beschriebenen Parameter.

Parameterbezeichnung	**alternative Bezeichnung**
-attributes=no	-header=no
-attributes=common	-
-attributes=all	-fileinfo, header=yes
-decompress	-uncompress
-flamcode	-character_set
-inrecdelim	-in_format=recdelim -irecdelim -irdelim -indelim
-inrecformat	-in_format -informat -irformat -irecformat
-inrecsize	-in_format=(Satzformat) -insize -irsize -irecsize
-maxbuffer	-buffer_size
-maxrecords	-records_in_buffer
-msgfile	-messages -message_file -log
-mode=cx7	-cx7 -seven-bit
-mode=cx8	-cx8 -eight-bit
-mode=vr8	-vr8

Parameterbezeichnung	alternative Bezeichnung
-option=cut	-cut
-option=nocut	-nocut
-outrecdelim	-out_format=recdelim -orecdelim -ordelim -outdelim
-outrecformat	-out_format -orecformat -orformat -outformat
-outrecsize	-out_format -orecsize -orsize -outsize
-parfile	-parameter_file
-show=no error attributes all	-info=no - -info=hold -info=yes
-translate	-code_table

FLAM (UNIX)

Benutzerhandbuch

Kapitel 3:

Der Unterprogramm-Aufruf flamup

3. Der Unterprogramm-Aufruf flamup

3.1 Aufruf-Sequenz für flamup

Die Anwendungsprogramme rufen das Unterprogramm flamup zur Komprimierung oder Dekomprimierung von einer oder mehreren Dateien, entsprechend den übergebenen Argumenten, auf.

Dieser Abschnitt beschreibt den Datentyp und die Semantik der Argumente. In konkreten Anwendungen, ist der Aufruf in die Syntax der jeweiligen Programmiersprache umzusetzen. In C folgen die Argumente dem Namen flamup, in runden Klammern eingeschlossen und durch Kommata getrennt.

flamup

Die Schnittstelle flamup stellt den Funktionsumfang des Shell-Kommandos FLAM Anwenderprogrammen zur Verfügung. Zusätzlich werden Schnittstellen für Benutzer-Ein-/Ausgabe und -Ausgänge untersützt.

Syntax

void **flamup** (char *******flamid*,
unsigned long **returncode*;
char **parameter_string*;
long **string_length*);

Argumente

flamid

Spezifiziert einen long integer (4 Bytes), der von FLAM während der Ausführung intern für die Datei-Identifikation benutzt wird.

returncode

Spezifiziert einen long integer (4 Bytes), der von flamup zur Rückgabe eines Returncodes an das aufrufende Programm benutzt wird.

parameter_string

Spezifiziert eine Zeichenkette, die die zu verwendenden FLAM-Parameter enthält (siehe Abschnitt 3.2 FLAM-Parameter im Unterprogramm Aufruf flamup). Die einzelnen FLAM-Parameter werden durch Kommata getrennt.

string_length

Spezifiziert einen long integer (4 Bytes), der die Länge des *parameter_string*-Arguments in Bytes enthält. Maximale Länge ist 512 Bytes.

Returncodes

Eine Beschreibung der Returncodes befindet sich im Anhang A, Returncodes.

Anwenderprogramme können die von flamup zurückgegebenen Returncodes gemäß den in Anhang A, Returncodes, beschriebenen Bedeutungen in Fehlermeldungen umsetzen. Anstelle einer eigenen Tabelle mit Fehlertexten kann auch die vom Kommando flam intern verwendete Message-Routine mit der FLAM-Tabelle benutzt werden.

Der Aufruf für die Ausgabe der Fehlermeldung mit den von FLAM intern verwendeten Tabellen lautet:

put_flmsg(returncode)

Die von FLAM verwendeten Definitionen, insbesondere auch die der symbolischen Namen für die verschiedenen Returncodes, sind in der C-Header Datei

/usr/include/flamincl.h

enthalten. Anwenderprogramme in anderen Programmiersprachen erfordern die Übertragung dieser Definitionen in die jeweilige Programmiersprache durch den Programmierer.

3.2 FLAM-Parameter im Unterprogramm-Aufruf flamup

Im Argument parameter_string des Unterprogrammaufrufs flamup können alle im Kommando flam gültigen Parameter mit den hierbei gültigen Werten stehen. Zu beachten ist die Abweichung bei Angabe von Listen von Dateinamen. Diese sind getrennt durch Komma "," und eingeschlossen in runde Klammern "(",")" beim jeweiligen Parameter anzugeben.

Beispiel:

FLAM...=(datei1,datei2,..datei*n*)

Neben den im Kommando flam gültigen Parametern können beim Unterprogrammaufruf die FLAM-Parameter für die Benutzer-Ein-/Ausgabe-Routinen (siehe auch Kapitel 5) und die Benutzerausgänge (siehe auch Kapitel 6) angegeben werden. Folgende Benutzer-Ein-/Ausgabe-Routinen können angegeben werden:

user_io

i(n)user_io

o(ut)user_io

Folgende Benutzerausgänge für Dateizugriffe können angegeben werden:

exk10

exk20

exd10

exd20

Die Benutzer- Ein-/Ausgabe-Routinen und Benutzerausgänge sind vom Anwender zu erstellen und müssen mit dem Anwenderprogramm und flamup (Modul flam_upu.o) gebunden werden.

Im folgenden Teil werden die nur in flamup zulässigen Parameter für die Benutzer-Ein/-Ausgabe-Routinen und Benutzerausgänge beschrieben.

exd10

Mit exd10 wird der Name eines Moduls angegeben, mit welchem Sätze der dekomprimierten Datei vor dem Schreiben bearbeitet werden.

Syntax

exd10=*Exitspezifikation*

Werte

Exitspezifikation

In dieser Version kann für *Exitspezifikation* nur exd10 angegeben werden.

Beschreibung

exd10 ist der Name eines Moduls, der die Sätze der dekomprimierten Datei(en) vor dem Schreiben bearbeitet. Dieser Modul muss vom Anwender erstellt und mit flamup und einem übergeordneten Anwenderprogramm gebunden werden (siehe Kapitel 7).

exd20

Mit exd20 wird der Name eines Moduls angegeben, der die Sätze der Komprimatsdatei nach dem Lesen bearbeitet.

Syntax

exd20=*Exitspezifikation*

Werte

Exitspezifikation

In dieser Version kann für *Exitspezifikation* nur exd20 angegeben werden.

Beschreibung

exd20 ist der Name eines Moduls, der die Sätze der Komprimatsdatei(en) nach dem Lesen bearbeitet. Dieser Modul muss vom Anwender erstellt und mit flamup und einem übergeordneten Anwenderprogramm gebunden werden (siehe Kapitel 7).

exk10

Mit exk10 wird der Name eines Moduls angegeben, mit welchem Sätze der Originaldatei nach dem Lesen bearbeitet werden

Syntax

exk10=*Exitspezifikation*

Werte

Exitspezifikation

In dieser Version kann für *Exitspezifikation* nur exk10 angegeben werden.

Beschreibung

Mit exk10 wird ein Modul definiert, der die Sätze der Originaldatei(en) nach dem Lesen bearbeitet. Dieser Modul muss vom Anwender erstellt und mit flamup und einem übergeordneten Anwenderprogramm gebunden werden (siehe Kapitel 7).

exk20

Mit exk20 wird der Name eines Moduls angegeben, mit welchem Sätze der Komprimatsdatei vor dem Schreiben bearbeitet werden.

Syntax

exk20=*Exitspezifikation*

Werte

Exitspezifikation

In dieser Version kann für *Exitspezifikation* nur exk20 angegeben werden.

Beschreibung

Mit exk20 wird ein Modul definiert, der die Sätze der Komprimatsdatei(en) vor dem Schreiben bearbeitet. Dieser Modul muss vom Anwender erstellt und mit flamup und einem übergeordneten Anwenderprogramm gebunden werden (siehe Kapitel 7).

inuser_io

Mit diesem Parameter wird angegeben, dass statt der Ein-/Ausgabe-Routinen von FLAM die des Benutzers für die Originaldatei benutzt werden sollen.

Syntax inuser_io

Werte keine

Beschreibung Die Originaldatei(en) werden mit den vom Anwender erstellten Ein-/Ausgaberoutinen gelesen. Dafür sind die Module usropn, usrget und usrcls erforderlich, die Module usrput und urspos können Dummy-Module sein (siehe Kapitel 6).

outuser_io

Mit diesem Parameter wird angegeben, dass statt der Ein-/Ausgabe-Routinen von FLAM die des Benutzers für die dekomprimierte Datei verwendet werden sollen.

Syntax outuser_io

Werte keine

Beschreibung Die dekomprimierte(n) Datei(en) werden mit den vom Anwender erstellten Ein-/Ausgaberoutinen geschrieben. Dafür sind die Module usropn, usrput und usrcls erforderlich, die Module usrget und urspos können Dummy-Module sein (siehe Kapitel 6).

user_io

Mit diesem Parameter wird angegeben, dass statt der Ein-/Ausgabe-Routinen von FLAM die des Benutzers für die Komprimatsdatei verwendet werden sollen.

Syntax

user_io

Werte

keine

Beschreibung

Die Komprimatsdatei(en) werden mit den vom Anwender erstellten Ein-/Ausgaberoutinen geschrieben oder gelesen. Erforderlich sind immer die Module usropn und usrcls, bei der Komprimierung usrput und bei der Dekomprimierung usrget. usrpos ist ein Dummy-Modul. Ein nicht erforderlicher Modul (usrget, usrput) kann ebenfalls ein Dummy-Modul sein (siehe Kapitel 6).

3.3 Binden von flamup

flamup liegt in zwei vorgebundenen Modulen vor, flam_up.o und flam_upu.o. Der Modul flam_up.o enthält Dummy-Module für die Benutzer-Ein-/Ausgabe-Routinen und die Benutzerausgänge. Der Modul flam_upu.o enthält keine Benutzer-Ein-/Ausgabe-Routinen und keine Benutzerausgänge.

flamup wird mit folgendem Kommando mit dem Anwenderprogramm (xyz) gebunden:

```
ld –o xyz /usr/lib/flame/flam_up.o  xyz.c –lc
```

Will der Anwender Benutzerausgänge und/oder eigene Ein-/Ausgaberoutinen verwenden, müssen diese zum Anwenderprogramm gebunden werden. In diesem Fall ist statt flam_up.o der Modul flam_upu.o zu verwenden.

Hierbei ist zu beachten, dass nicht nur einzelne Routinen vom Anwender zu erstellen sind, sondern alle Benutzerausgänge und/oder Ein-/Ausgabe-Routinen. Will der Anwender nur eigene Benutzerausgänge oder nur eigene Ein-/Ausgabe-Routinen verwenden, kann er für die übrigen Funktionen die in der /usr/lib/flame vorhandenen Dummy-Routinen flam_usrmod.o bzw. flam_exitmod.o einbinden.

Das Kommando lautet dann:

```
ld –o xyz /usr/lib/flame/flam_upu.o
/usr/lib/flame/flam_usrmod.o
/usr/lib/flame/flam_exitmod.o xyz.c –lc
```

/usr/lib/flame/flam_usrmod.o und/oder

/usr/lib/flame/flam_exitmod.o sind durch Module des Anwenders zu ersetzen.

Bei einigen UNIX-Systemen muss die Runtime-library /lib/crt0.o im Link-Kommando mit angegeben werden, z.B. SCO-UNIX.

FLAM (UNIX)

Benutzerhandbuch

Kapitel 4:

Die Satzschnittstelle flamrec

4. Die Satzschnittstelle flamrec

4.1 Funktionen der Satzschnittstelle

Die FLAM-Satzschnittstelle flamrec besteht aus einer Reihe von Unterprogrammen, die von allen Programmiersprachen, in denen C-Funktionen benutzt werden, aufgerufen werden können. Bis auf die Schlüsselbeschreibung, die in einer späteren Version benutzt wird, sind alle Argumente durch elementare Datentypen (long integer, string) dargestellt. Die Übergabe der Argumente erfolgt durch Pointer, nicht als Werte. Die Anmerkungen zur Notation in Abschnitt 1.5, Schnittstellen, gelten auch hier.

Anwendungsprogramme, die in C geschrieben werden, können die Anweisung #include flamincl.h enthalten. Diese C-Header-Datei enthält die Definitionen der symbolischen Konstanten und Returncodes sowie die Struktur für die Schlüsselbeschreibung. Für Programme in anderen Programmiersprachen sind diese Definitionen sinngemäß zu übertragen.

Alle Argumentlisten beginnen einheitlich mit einer Kennung, die zur eindeutigen Identifikation der Komprimatsdatei dient. Dieses Argument flmid enthält die von flmopn eingetragene Adresse des Arbeitsbereichs für die jeweilige Komprimatsdatei und darf bis zum flmcls nicht verändert werden. Alle übrigen Argumente sind nur für die Funktion von Interesse, der sie übergeben werden.

Auch das Argument returncode ist in jeder Argumentliste enthalten und dient zur Rückmeldung der erfolgreichen Durchführung der jeweiligen Funktion bzw. eines möglichen Fehlers. Alle Codes und ihre Bedeutungen sind in Anhang A, Returncodes, aufgelistet, ebenso die Bedeutung der zusätzlichen Informationen im höchstwertigen Byte des Returncodes in allen Fehlerfällen, die von Dateizugriffen herrühren. Wie beim Unterprogramm flamup können auch die Returncodes der Funktionen der Satzschnittstelle in Meldungsnummern umgesetzt oder mit der von FLAM intern verwendeten Message-Routine put_flmsg ausgegeben werden.

Im Einzelnen umfasst die Satzschnittstelle folgende Funktionen:

Funktion	**Zweck**	**Entspricht I/O Aufrufen**
flmcls	close: Verarbeitung der Originaldatei abschließen und die FLAMFILE schließen	close
flmflu	flush: Verarbeitung der Originaldatei abschließen, ohne die FLAMFILE zu schließen	fflush
flmget	get: Einen Originalsatz sequentiell lesen	(f)gets/ read
flmghd	get header: Den allgemeinen FLAM-Fileheader lesen	
flmguh	get user header: Den anwenderspezifischen FLAM-Fileheader lesen	
Flmopn, flmopd, flmopf	open: FLAMFILE öffnen	create/ (f)open
flmphd	put header: Den allgemeinen FLAM-Fileheader schreiben	
flmpos	position: Auf den nächsten Fileheader positionieren	
flmpuh	put user header: Den anwenderspezifischen FLAM-Fileheader schreiben	
flmput	put: Einen Originalsatz sequentiell schreiben	(f)puts/ write
flmpwd	password: Kennwort übergeben	

4.2 Programmierung der Satzschnittstelle bei der Komprimierung

Die Erzeugung einer FLAMFILE verläuft in drei Phasen:

- der Open-Sequenz

- einem oder mehreren Komprimierungszyklen

- dem Abschluss (Funktion **flmcls**)

Die Open-Sequenz beginnt immer mit dem Aufruf der Funktion **flmopn**. Im Anschluss daran können die Funktionen **flmopd** und/oder **flmopf** aufgerufen werden, wenn in **flmopn** der Parameter continue_param gesetzt wurde. Bei Aufruf beider Funktionen muss **flmopd** zuerst erfolgen und auch hier der Parameter continue_param gesetzt sein.

Werden **flmopd** oder **flmopf** von der Anwendung nicht aufgerufen, erzeugt FLAM intern entsprechende Aufrufe mit den impliziten Einstellungen. Die für das flam-Kommando und das Unterprogramm flamup einstellbaren Defaultwerte sind hier nicht wirksam.

Schließlich muss, falls verschlüsselt werden soll, mit dem Aufruf von **flmpwd** das Kennwort übergeben werden.

In der zweiten Phase der Verarbeitung wird für jede zu komprimierende Originaldatei ein Komprimierungszyklus durchlaufen.

In der Regel muss ein solcher Zyklus mit einem **flmphd**-Aufruf beginnen, wodurch ein allgemeiner FLAM-Fileheader erzeugt wird. Lediglich wenn eine einzelne Originaldatei ohne Verschlüsselung komprimiert wird, ist dieser Aufruf optional.

Wurde ein FLAM-Fileheader erzeugt, kann durch Aufruf von **flmpuh** ein anwenderspezifischer Header angehängt werden. Bei Secure FLAMFILEs ist dieser Aufruf obligatorisch, ggf. mit Headerlänge 0.

Nun können Daten zur Komprimierung übergeben werden durch wiederholte Aufrufe von **flmput**. Jeder Aufruf von **flmput** übergibt einen Satz im Sinne von FLAM, der auf Grund der von FLAM ebenfalls gespeicherten Strukturinformation bei der Dekomprimierung genau so oder auch mit anderen Satzattributen wiederhergestellt werden kann. FLAM akkumuliert die übergebenen Daten im Komprimierungspuffer und steuert ohne Zutun der Anwendung die Komprimierung und die Ausgabe des Komprimats.

Zum Abschluss des Komprimierungszyklus muss **flmflu** aufgerufen werden, um möglicherweise noch im Puffer befindliche Sätze zu komprimieren und deren Komprimat

auszugeben. Dabei gibt FLAM statistische Informationen an die Anwendung zurück.

Jetzt kann entweder ein neuer Komprimierungszyklus durch Aufruf von **flmphd** begonnen oder durch Aufruf von **flmcls** die FLAMFILE geschlossen werden.

Das aufrufende Programm erhält immer die Kontrolle zurück. Es gibt keine Fehlerausgänge, und es werden von der Satzschnittstelle auch keine Fehlermeldungen erzeugt, sondern Returncodes an die Anwendung zurückgegeben.

4.3 Programmierung der Satzschnittstelle bei der Dekomprimierung

Die Dekomprimierung einer FLAMFILE gliedert sich analog zur Komprimierung in die Phasen

- Open-Sequenz

- Dekomprimierungszyklen

- Abschluss (Funktion **flmcls**)

Hierbei besteht die Open-Sequenz aus den gleichen Aufrufen wir bei der Komprimierung, lediglich die Übergaberichtung einiger Argumente - beispielsweise des Komprimierungsmodus - ist umgekehrt, d.h. sie werden von FLAM an die Anwendung zurückgegeben.

Im Dekomprimierungszyklus gibt es für jede Funktion des Komprimierungszyklus ein Pendant, die die komplementäre Aufgabe erfüllt. Zu **flmphd** und **flmpuh** sind die Komplemente **flmghd** bzw. **flmguh**, zu **flmput** ist es **flmget**, und **flmflu** wird auch für den Abschluss des Dekomprimierungszyklus benutzt. Darüberhinaus gibt es für die Dekomprimierung die Funktion **flmpos**, mit der auf den Anfang des Komprimats der nächsten Originaldatei positioniert werden kann und zu der es keine Entsprechung bei der Komprimierung gibt.

Typischerweise beginnt ein Dekomprimierungszyklus mit einem **flmghd**-Aufruf, aus dem sich die Information über die Originaldatei wie deren Name, Satzformat etc. ergibt. Ggf. folgt ein **flmguh**, falls die FLAMFILE einen anwenderspezifischen Fileheader enthält. Danach können mehrfach **flmget**-Aufrufe folgen, bei denen die Anwendung jeweils einen Satz der Originaldatei erhält. Das Ende einer Originaldatei wird durch einen entsprechenden Returncode des **flmget**-Aufrufs signalisiert, kann aber auch gezielt durch einen **flmpos**-Aufruf direkt erreicht werden. Mit einem **flmflu** wird ein Dekomprimierungszyklus beendet, dem ein weiterer Zyklus oder der Abschluss mit **flmcls** folgen kann. Ein Aufruf von **flmflu** oder **flmpos** je Zyklus ist obligatorisch, die übrigen Aufrufe sind optional.

4.4 Beschreibung der flamrec-Funktionen

Im folgenden Teil sind die Funktionen der Satzschnittstelle in alphabetischer Reihenfolge beschrieben.

flmcls

Mit der Funktion flmcls (close) wird der Zugriff auf die Satzschnittstelle beendet. Bei der Komprimierung wird noch die letzte Matrix komprimiert, das Komprimat auf die FLAMFILE geschrieben und dann die FLAMFILE geschlossen. Bei Secure FLAMFILEs wird auch ein Filetrailer erzeugt.

Beim Dekomprimieren wird nur die FLAMFILE geschlossen; falls noch vorhanden, werden restliche Originalsätze nicht mehr übergeben.

Falls beim flmopn angefordert (*statistics* ≠ 0), werden die Statistikinformationen mit übergeben.

Syntax

void **flmcls** (char **_flmid_,
long *_returncode_,
unsigned long *_cputime_,
unsigned long *_records_,
unsigned long *_bytes_,
unsigned long *_byteofl_,
unsigned long *_cmprecs_,
unsigned long *_cmpbytes_,
unsigned long *_cmpbytofl_);

Argumente

flmid
Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode
In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

cputime
In diesem Argument wird die aktuelle Zeit in Einheiten von Sekunden zurückgegeben.

records
Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl der mit flmput übergebenen bzw. mit flmget übernommenen dekomprimierten Sätze zurückgegeben. Die in diesem Zähler kumulierten Werte sind nur bei Verarbeitung vollständiger Dateien signifikant.

bytes
Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl der Bytes in den mit flmput übergebenen bzw. mit flmget übernommenen dekomprimierten Sätze zurückgegeben. Die in diesem Zähler kumulierten Werte sind nur bei Verarbeitung vollständiger Dateien signifikant.

byteofl — Übersteigt der im Argument bytes kumulierte Wert 2.000.000.000, so wird dieser Zähler für die Vielfachen von 2.000.000.000 benutzt.

cmprecs — Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* $\neq 0$), wird in diesem Argument die Anzahl bei der Komprimierung erzeugter bzw. bei der Dekomprimierung gelesener Komprimatssätze zurückgegeben. Dieser Wert ist nur bei der Verarbeitung vollständiger Dateien signifikant.

cmpbytes — Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* $\neq 0$), wird in diesem Argument die Anzahl bei der Komprimierung erzeugter bzw. bei der Dekomprimierung gelesener Bytes zurückgegeben. Dieser Wert ist nur bei der Verarbeitung vollständiger Dateien signifikant.

cmpbytofl — Übersteigt der im Argument cmpbytes kumulierte Wert 2.000.000.000, so wird dieser Zähler für die Vielfachen von 2.000.000.000 benutzt.

Abhängigkeiten — Bei Secure FLAMFILEs darf flmcls nur unmittelbar nach flmflu oder flmpos aufgerufen werden.

Returncodes — Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmflu

Mit flmflu (flush) wird die Komprimierung einer Datei beendet; es werden die restlichen Sätze im Block komprimiert und geschrieben, erforderlichenfalls unter Auffüllung des letzten FLAMSatzes. Die Komprimatsdatei wird nicht geschlossen. Die Zähler für die Statistik werden nach flmflu nicht zurückgesetzt. Bei Secure FLAMFILEs wird bei der Komprimierung ein Membertrailer erzeugt.

Syntax

void **flmflu** (char ***flmid*,
long **returncode*,
unsigned long **cputime*,
unsigned long **records*,
unsigned long **bytes*,
unsigned long **byteofl*,
unsigned long **cmprecs*,
unsigned long **cmpbytes*,
unsigned long **cmpbytofl*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

cputime

In diesem Argument wird die aktuelle Zeit in Einheiten von Sekunden zurückgegeben.

records

Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl der mit flmput übergebenen bzw. mit flmget übernommenen dekomprimierten Sätze zurückgegeben. Die in diesem Zähler kumulierten Werte sind nur bei Verarbeitung vollständiger Dateien signifikant.

bytes

Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl der Bytes in den mit flmput übergebenen bzw. mit flmget übernommenen dekomprimierten Sätze zurückgegeben. Die in diesem Zähler kumulierten Werte sind nur bei Verarbeitung vollständiger Dateien signifikant.

byteofl

Übersteigt der im Argument bytes kumulierte Wert 2.000.000.000, so wird dieser Zähler für die Vielfachen von 2.000.000.000 benutzt.

cmprecs

Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl bei der Komprimierung erzeugter bzw. bei der Dekomprimierung gelesener Komprimatssätze zurückgegeben. Dieser Wert ist nur bei der Verarbeitung vollständiger Dateien signifikant.

cmpbytes

Falls mit flmopn die Statistik eingeschaltet wurde (*statistics* ≠ 0), wird in diesem Argument die Anzahl bei der Komprimierung erzeugter bzw. bei der Dekomprimierung gelesener Bytes zurückgegeben. Dieser Wert ist nur bei der Verarbeitung vollständiger Dateien signifikant.

cmpbytofl

Übersteigt der im Argument cmpbytes kumulierte Wert 2.000.000.000, so wird dieser Zähler für die Vielfachen von 2.000.000.000 benutzt.

Abhängigkeiten

Beim Komprimieren von Secure FLAMFILES darf flmflu nur nach flmpuh oder flmput ausgeführt werden. Beim Dekomprimieren von Secure FLAMFILES muss vor flmflu ein flmget, flmghd oder flmguh aufgerufen werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmget

Mit der Funktion flmget (get record) wird der jeweils nächste Originalsatz in sequentieller Folge gelesen. Die Daten werden dabei in den Satzpuffer des aufrufenden Programms übertragen. Die Erkennung eines Fileheaders wird ggf. im Returncode signalisiert.

Syntax

void **flmget** (char ***flmid*,
long **returncode*,
long **record_length,*
char **record*,
long **buffer_length*);

Argumente

flmid
Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode
In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

record_length
In diesem Argument wird die Länge des gelesenen Satzes in Bytes zurückgegeben.

record
Dieses Argument ist der Satzpuffer.

buffer_length
Dieses Argument enthält die Länge des Satzpuffers in Bytes.

Abhängigkeiten

flmget darf nur beim Dekomprimieren benutzt werden. Bei verschlüsselten FLAMFILEs muss vor dem ersten Aufruf von flmget das Kennwort mit flmpwd übergeben werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmghd

Mit flmghd (get file header) können bei der Dekomprimierung die Attribute der Originaldatei angefordert werden, sofern diese in der FLAMFILE gespeichert sind.

Sind in der FLAMFILE mehrere Fileheader vorhanden (siehe flmphd), so wird mit flmghd jeweils der letzte von FLAM erkannte Fileheader übergeben.

Syntax

void **flmghd** (char ***flmid*,
long **returncode*,
long **filename_length*,
char **filename*,
long **organization*,
long **record_format*,
long **record_size*,
char **record_delimiter*,
struct kd **key_description*,
long **block_size*,
long **print_control*,
char **system*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

filename_length

Dieses Argument enthält die Länge des Pufferbereichs, in dem der Name der Originaldatei zurückzugeben ist. Hier wird auch die tatsächliche Länge zurückgegeben.

filename

Dieses Argument gibt den Pufferbereich an, in dem der Name der Originaldatei zurückzugeben ist. Der Name wird dort als Zeichenkette abgelegt. Ist die Bereichslänge nicht ausreichend, werden nur so viele Zeichen des Namens zurückgegeben, wie der Bereich fassen kann.

organization

In diesem Argument wird ein Wert für die Dateiorganisation der Originaldatei zurückgegeben. Die möglichen Werte und ihre Bedeutungen, die teilweise Organisationsformen kennzeichnen, die unter UNIX nicht existieren bzw. noch nicht unterstützt werden, sind:

0 sequentiell

1	indexsequentiell
2	relativ
3	Direktzugriff
4	keine Satzstruktur
5	Bibliothek
6	physikalisch

record_format

In diesem Argument wird ein Wert für das Satzformat der Originaldatei zurückgegeben. Die möglichen Werte und ihre Bedeutungen, die teilweise Satzformate kennzeichnen, die unter UNIX nicht existieren, sind:

0	variable
1	fixed
2	undefined
3	stream
8	var/blocked
9	fix/blocked
16	var/blocked/spanned
17	fix/blocked/Standard
18	eaf
24	vfc
32	var_4b
40	var_ascii
48	var_ebcdic

record_size

In diesem Argument wird die Satzlänge der Originaldatei zurückgegeben. Der Wert 0 wird bei variablen Satzlängen und bei Satzformat stream gesetzt.

record_delimiter

In diesem Argument wird bei Satzformat stream das Satztrennzeichen angegeben. Die Länge dieses Character Strings ist 4 Byte (entsprechend einem long integer).

record_delimiter hat folgenden Aufbau:

Im höchstwertigen Byte steht die Anzahl der Bytes. Bei Anzahl der Bytes=1 steht im niedrigstwertigen Byte das Satztrennzeichen. Bei Anzahl Bytes=2 steht im zweitniedrigstwertigen Byte das 1., im niedrigstwertigen Byte das 2. Byte des Satztrennzeichens, z.B.:

0x0100000a Satztrennzeichen LF (0x0a)
0x02000d0a Satztrennzeichen CR/LF (0x0d0a)

key_description

Dieses Argument ist für künftige Erweiterungen reserviert

block_size

Dieses Argument ist für künftige Erweiterungen reserviert.

print_control

Dieses Argument ist für künftige Erweiterungen reserviert.

system

Dieses Argument besteht aus zwei Bytes, in denen ein Code für das Betriebssystem zurückgegeben wird, auf dem die FLAMFILE erzeugt wurde. Dieser hexadezimale Code hat folgende Bedeutung:

00 00 unbekanntes System
00 0E MS-Windows
01 01 IBM MVS
01 02 IBM VSE
01 03 IBM VM
03 01 HP OpenVMS
09 01 HP NonStop
12 04 Unix

Abhängigkeiten

flmghd darf nur beim Dekomprimieren benutzt werden. Es kann aufgerufen werden, um die Dateiattribute des ersten Members einer FLAMFILE anzufordern, wenn FLAM nach dem Öffnen (flmopn / flmopd / flmopf) den Returncode 1 zurückgibt. Die Attribute eines Folgemembers sind verfügbar, wenn ein vorangehender flmget-Aufruf den Returncode 6 zurückgibt.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmguh

Die Funktion flmguh (get user header) dient zur Wiedergewinnung der Daten aus dem anwenderspezifischen FLAM-Fileheader bei der Dekomprimierung. Sie ist nur bei der Dekomprimierung und unmittelbar nach einem Aufruf von flmghd zulässig.

Syntax

void **flmguh** (char ***flmid*,
long **returncode*,
long **user_attributes_length*,
char **user_attributes*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

user_attributes

In diesem Argument wird die Länge des Speicherbereichs übergeben, in dem der anwenderspezifische FLAM-Fileheader abzulegen ist. Die effektiv benutzte Länge wird zurückgegeben.

user_attributes

In diesem Argument wird der anwenderspezifische FLAM-Fileheader zurückgegeben.

Abhängigkeiten

flmguh darf nur beim Dekomprimieren benutzt werden. Es gibt die Daten aus dem anwenderspezifischen Fileheader zurück, die dort mit flmpuh gespeichert wurden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmopd

Mit der Funktion flmopd (open/Dateibeschreibung) können spezielle Dateieigenschaften der FLAMFILE explizit gesetzt oder abgefragt werden. Dieser Aufruf ist nicht in jedem Fall erforderlich. Wird er bei der Komprimierung nicht ausgeführt, werden für die betreffenden Dateieigenschaften der FLAMFILE implizite Werte eingesetzt. Diese sind ggf. jeweils bei den Beschreibungen der betref- fenden Argumente angegeben.

flmopd kann nur im Anschluss an flmopn aufgerufen werden.

Syntax

void **flmopd** (char ***flmid*,
long **returncode*,
long **continue_param*,
long **filename_length*,
char **filename*,
long **organization*,
long **record_format*,
long **record_size*,
char **record_delimiter*,
struct kd **key_description*,
long **block_size*,
long **close_disposition*,
long **device*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

continue_param

Mit diesem Argument wird angezeigt, ob anschließend die Übergabe bzw. Übernahme weiterer Einstellungsdaten durch einen flmopf-Aufruf erfolgen soll oder nicht.

0	kein flmopf-Aufruf
sonst	flmopf-Aufruf folgt

filename_length

In diesem Argument wird die Länge des Bereichs für den Namen der zu öffnenden FLAMFILE übergeben und die tatsächliche Länge des Namens zurückgegeben.

filename Dieses Argument ist der Bereich, in dem der Name der zu öffnenden FLAMFILE zurückgegeben wird.

organization In diesem Argument wird ein Code für die Dateiorganisation der zu öffnenden FLAMFILE übergeben bzw. zurückgegeben.

Die möglichen Werte sind:

0 sequentiell

Impliziter Wert (bei der Komprimierung): 0.

record_format In diesem Argument wird ein Code für das Satzformat der zu öffnenden FLAMFILE übergeben bzw. zurückgegeben.

Die möglichen Werte sind:

0 var bei mode=adc/cx8/vr8
1 fixed bei mode=adc/cx8/vr8
3 stream (nur bei mode=cx7)

Impliziter Wert (bei der Komprimierung): 1.

record_size In diesem Argument wird die maximale Satzlänge der zu öffnenden FLAMFILE übergeben bzw. zurückgegeben.

Impliziter Wert (bei der Komprimierung): 512.

record_delimiter Die Länge dieses Character Strings ist 4 Byte (entsprechend einem long integer).

Der Aufbau des record_delimiter ist bei Funktion flmghd beschrieben.

key_description Dieses Argument ist für künftige Erweiterungen reserviert.

block_size Dieses Argument ist für künftige Erweiterungen reserviert.

close_disposition Dieses Argument ist für künftige Erweiterungen reserviert.

device Dieses Argument dient zur Aktivierung der benutzereigenen Ein-/Ausgabe. Diese Routinen müssen anstelle der Ein-/Ausgabe-Befehle zur Verfügung gestellt und in das Anwenderprogramm eingebunden werden.

7 Benutzer-Ein-/Ausgabe
sonst Standard

Impliziter Wert (bei der Komprimierung): 0.

Abhängigkeiten

flmopd ist nur nach einem flmopf-Aufruf mit *continue_param* ≠ 0 zulässig. Auch flmopd selbst muss mit *continue_param* ≠ 0 aufgerufen werden, wenn danach flmopf aufgerufen werden soll.

Bei Aufruf mit *device*=7 muss die Anwendung mit den Benutzer-Ein-/Ausgaberoutinen gemäß Abschnitt 5.2 gebunden werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmopf

Mit der Funktion flmopf (open FLAM-Einstellungen) können die FLAM-Einstellungen explizit gesetzt oder abgefragt werden. Dieser Aufruf ist nicht in jedem Fall erforderlich. Wird er bei der Komprimierung nicht ausgeführt, werden für die betreffenden Einstellungen implizite Werte eingesetzt. Diese sind ggf. jeweils bei den Beschreibungen der betreffenden Argumente angegeben.

flmopf kann nur als letzter Funktionsaufruf der Open-Sequenz benutzt werden, also nur nach flmopd, falls diese aufgerufen wird, oder nach flmopn, falls kein flmopd-Aufruf erfolgt.

Syntax

void **flmopf** (char ***flmid*,
long **returncode*,
long **flam_version*,
long **flam_code*,
long **comp_mode*,
long **max_buffer*,
long **fileheader*,
long **max_records*,
struct kd **key_description*,
long **block_mode*,
char **exitroutine_comp*,
char **exitroutine_decomp*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

flam_version

Dieses Argument muss immer den Wert 2 enthalten.

flam_code

In diesem Argument wird beim Komprimieren angegeben, in welchem Zeichencode zeichencodierte Informationen der FLAMFILE zu erzeugen sind. Bei der Dekomprimierung wird diese Information zurückgegeben.

0 EBCDIC
1 ASCII

Impliziter Wert (bei der Komprimierung): 1.

comp_mode

In diesem Argument wird beim Komprimieren angegeben, welches Kompressions- bzw. Verschlüsselungsverfahren zu verwenden ist. Bei der Dekomprimierung wird diese Information zurückgegeben.

Folgende hexadezimale (dezimale) Werte sind möglich:

Kompressi-ons-verfahren	**Verschlüsselungsverfahren**		
	ohne	**FLAMenc**	**AES**
cx8	0x00 (0)	nicht zulässig	nicht zulässig
cx7	0x01 (1)	nicht zulässig	nicht zulässig
vr8	0x02 (2)	nicht zulässig	nicht zulässig
adc	0x03 (3)	0x13 (19)	0x23 (35)
ndc	0x0B (11)	0x1B (27)	0x2B (43)

Impliziter Wert (bei der Komprimierung): 0.

max_buffer

In diesem Argument wird beim Komprimieren die zu benutzende Puffergröße vorgegeben. Bei der Dekomprimierung wird diese Information als Anzahl Bytes zurückgegeben.

Die Puffergröße wird in Bytes oder KBytes vorgegeben. Gültig sind alle Werte zwischen 1 und 2.621.440. Details über die Interpretation dieser Angabe und die tatsächlich benutzten Puffergrößen finden sich bei der Beschreibung des Parameters maxbuffer im Abschnitt "Das Kommando flam".

Impliziter Wert (bei der Komprimierung): 32 Kbytes.

fileheader

In diesem Argument wird bei der Dekomprimierung die Information zurückgegeben, ob die FLAMFILE einen FLAM-Fileheader enthält oder nicht. Hiervon hängt ab, ob ein anschließender flmghd-Aufruf sinnvoll ist.

Bei der Komprimierung wird dieses Argument nicht benutzt.

0 kein FLAM-Fileheader
1 FLAM-Fileheader vorhanden

Impliziter Wert (bei der Komprimierung): 0.

max_records In diesem Argument wird beim Komprimieren die zu benutzende Satzanzahl pro Matrix vorgegeben. Bei der Dekomprimierung wird diese Information zurückgegeben.

Impliziter Wert (bei der Komprimierung): 255.

key_description Dieses Argument gibt den Speicherbereich an, in dem für eine indexsequentielle Originaldatei die Beschreibung des Schlüssels übergeben bzw. zurückgegeben wird.
Dieses Argument ist für künftige Erweiterungen reserviert.

block_mode Dieses Argument ist für künftige Erweiterungen reserviert.

exitroutine_comp Dieses Argument enthält den Namen eines Benutzerausgangs, also eines ausführbaren Programms, das beim Komprimieren bei jedem FLAMFILE-Zugriff aufgerufen wird. Dieses Programm muss als C-Funktion mit dem Namen exk20 mit in das Anwenderprogramm eingebunden sein. Der Aufruf des Benutzerausgangs unterbleibt, wenn der Name mit einer Leerstelle (X'20') oder einer Binärnull (X'00') beginnt.

Falls der Benutzerausgang exitroutine_comp den Komprimatssatz modifiziert, muss beim Dekomprimieren ein entsprechender Benutzerausgang *exitroutine_decomp* angegeben werden, durch den die Änderung vor der Verarbeitung des Satzes durch FLAM rückgängig gemacht wird.

exitroutine_decomp Dieses Argument enthält den Namen eines Benutzerausgangs, also eines ausführbaren Programms, das beim Dekomprimieren bei jedem FLAMFILE-Zugriff aufgerufen wird. Dieses Programm muss als C-Funktion mit dem Namen exd20 mit in das Anwenderprogramm eingebunden sein.

Der Aufruf des Benutzerausgangs unterbleibt, wenn der Name mit einer Leerstelle (X'20') oder einer Binärnull (X'00') beginnt.

Ein Benutzerausgang exitroutine_decomp muss angegeben werden, falls beim Komprimieren Sätze der FLAMFILE durch einen Benutzerausgang *exitroutine_comp* verändert worden sind, und alle Änderungen müssen in diesem Benutzerausgang rückgängig gemacht werden.

Abhängigkeiten keine

Returncodes Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmopn

Die Funktion flmopn (open) leitet die Open-Sequenz für eine FLAMFILE ein. Ihrem Aufruf können Aufrufe von flmopd und/oder flmopf folgen, um Dateieigenschaften der FLAMFILE bzw. FLAM-Einstellungen zu setzen oder abzufragen.

Syntax

void **flmopn** (char ***flmid*;
long **returncode*;
long **continue_param*;
long **flam_open*;
char **filename*;
long **statistics*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

continue_param

Mit diesem Argument wird angezeigt, ob anschließend die Übergabe bzw. Übernahme weiterer Einstellungsdaten durch einen flmopd- oder einen flmopf-Aufruf erfolgen soll oder nicht.

0	kein weiterer Aufruf
sonst	weiterer Aufruf folgt

flam_open

Dieses Argument gibt den Verarbeitungsmodus der FLAMFILE an.

0	input (FLAMFILE lesen, d.h. dekomprimieren)
1	output (FLAMFILE schreiben, d.h. komprimieren)

filename

In diesem Argument wird der Dateiname für die FLAMFILE übergeben. Diese Zeichenkette muss mit einem Null-Byte (Binärnull) abgeschlossen werden. Der Dateiname wird durch die Funktion flmopd zurückgegeben.
Bei Verwendung von stdin/stdout steht im 1.Byte eine binäre Null oder ein Leerzeichen.

statistics

In diesem Argument wird angegeben, ob mit Abschluss der Komprimierung Statistikinformationen gewünscht werden oder nicht.

0	keine Statistik
sonst	Statistikinformationen zurückgeben

Abhängigkeiten

Wird auf die Folgeaufrufe flmopd und/oder flmopf bei der Komprimierung verzichtet (*continue_param*=0), so öffnet flmopn die FLAMFILE unter Verwendung impliziter Werte für die entsprechenden Eigenschaften und Einstellungen.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmphd

Die Funktion flmphd (put file header) ist nur bei der Komprimierung zugelassen. Sie erzeugt einen allgemeinen Fileheader, der das Dateiformat der anschließend übergebenen Originalsätze beschreibt. Werden mehrere Dateien in eine FLAMFILE komprimiert, so kann für jede Datei ein Fileheader mit der Funktion flmphd erzeugt werden.

FLAM gibt diese Fileheaderinformationen auf Anforderung (flmghd) beim Dekomprimieren zurück.

Syntax

void **flmphd** (char ***flmid*;
long **returncode*;
long **filename_length*;
char **filename*;
long **organization*;
long **record_format*;
long **record_size*;
char **record_delimiter*;
struct kd **key_description*;
long **block_size*;
long **print_control*;
char **system*;
long **continue_param*;

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

filename_length

Dieses Argument enthält die Länge des Pufferbereichs, in dem der Name der Originaldatei übergeben wird.

filename

Dieses Argument gibt den Pufferbereich an, in dem der Name der Originaldatei übergeben wird. Der Name wird dort als Zeichenkette abgelegt und mit '\0' beendet.

organization

In diesem Argument wird ein Code für die Dateiorganisation der Originaldatei übergeben.

Die zulässigen Werte sind bei der Funktion flmghd beschrieben.

record_format In diesem Argument wird ein Code für das Satzformat der Originaldatei übergeben.

Die zulässigen Werte sind bei der Funktion flmghd beschrieben.

record_size In diesem Argument wird die maximale Satzlänge der Originaldatei übergeben. Die zulässigen Werte sind bei der Funktion flmghd beschrieben.

record_delimiter In diesem Argument wird bei Satzformat stream das Satztrennzeichen abgelegt. Die Länge dieses Character Strings ist 4 Byte (entsprechend einem long integer).

Der Aufbau des record_delimiter ist bei Funktion flmghd beschrieben.

key_description Dieses Argument gibt den Speicherbereich an, in dem für eine indexsequentielle Originaldatei die Beschreibung des Schlüssels übergeben wird.

Dieses Argument ist für künftige Erweiterungen reserviert.

print_control Dieses Argument ist für künftige Erweiterungen reserviert.

system Dieses Argument besteht aus zwei Bytes, in denen ein Code für das Betriebssystem übergeben wird, auf dem die FLAMFILE erzeugt wird.

Die Liste der gültigen Werte ist bei flmghd aufgeführt.

continue_param Mit diesem Argument wird angezeigt, ob anschließend die Übergabe weiterer Informationen für den FLAM-Fileheader durch flmpuh erfolgen soll oder nicht.

0	keine weiteren Informationen
sonst	weitere Informationen folgen

Abhängigkeiten Die Funktion flmphd darf nur beim Komprimieren benutzt werden. Sie ist nur erlaubt, wenn bei flmopf *fileheader*=1 angegeben wird. Bei Aufruf mit *continue_param* ≠ 0 oder nach flmpwd muss ein flmpuh-Aufruf unmittelbar folgen.

Returncodes Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmpos

Mit flmpos (position) kann beim Dekomprimieren bis zum Ende der Daten einer Originaldatei positioniert werden, um den Fileheader der nächsten Datei zu lesen.

Syntax

void **flmpos** (char ***flmid*;
long **returncode*;
long **position*;

Argumente

flmid
Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode
In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

position
In diesem Argument wird beim Dekomprimieren mit 99.999.998 auf das Ende der Daten einer Originaldatei positioniert, also auf den nächsten FLAM-Fileheader bzw. das Dateiende der FLAMFILE.

Abhängigkeiten

flmpos darf nur beim Dekomprimieren benutzt werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmpuh

Mit der Funktion flmpuh (put user header) können in der FLAMFILE weitere Informationen beliebiger Struktur gespeichert werden. Diese werden als Zeichenkette eingefügt und können bei der Dekomprimierung mit der Komplementärfunktion flmguh verfügbar gemacht werden.

Die Funktion flmpuh ist nur bei der Komprimierung und nur unmittelbar nach einem Aufruf von flmphd zulässig.

Syntax

void **flmpuh** (char ***flmid*;
long **returncode*;
long **user_attr_length*;
char **user_attributes*;

Argumente

flmid
Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode
In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

user_attr_length
In diesem Argument wird die Länge des anwenderspezifischen Headers übergeben (max. Länge 32.732 Bytes).

user_attributes
Dieses Argument gibt einen Speicherbereich an, der den anwenderspezifischen Header enthält, dessen Inhalt vom Anwender beliebig gesetzt werden kann.

Abhängigkeiten

flmpuh darf nur beim Komprimieren benutzt werden. Es darf nur unmittelbar nach flmphd aufgerufen werden und ist bei Secure FLAMFILEs obligatorisch.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmput

Mit der Funktion flmput (put record) wird jeweils ein Originalsatz zum Komprimieren übergeben.

Syntax

void **flmput** (char ***flmid*,
long **returncode*,
long **record_length*,
char **record*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

record_length

Dieses Argument enthält die Satzlänge in Bytes.

record

Dieses Argument ist der Satzpuffer.

Abhängigkeiten

flmput darf nur beim Komprimieren benutzt werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

flmpwd

Mit der Funktion flmpwd (password) wird beim Komprimieren ein Kennwort zum Ver-, beim Dekomprimieren zum Entschlüsseln der Komprimatsdaten übergeben.

Syntax

void **flmpwd** (char ***flmid*;
long **returncode*;
long **pwd_length*;
char **pwd*);

Argumente

flmid

Der Feldinhalt von *flmid* ist ein Pointer auf einen Character String. Dieses Argument identifiziert die FLAMFILE, für die die Funktion auszuführen ist. Es wird durch die Funktion flmopn gesetzt und darf bis zum flmcls nicht verändert werden.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

pwd_length

Dieses Argument enthält die Länge des Kennworts in Bytes.

pwd

Dieses Argument ist der Puffer, der das Kennwort enthält.

Abhängigkeiten

flmpwd darf nur bei Ver-/Entschlüsselung verwendet werden.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

4.5 Einbinden von Funktionen der Satzschnittstelle in Anwenderprogramme

Die Satzschnittstelle von FLAM liegt in zwei vorgebundenen Modulen vor, flam_rec.o und flam_recu.o. Der Modul flamrec.o enthält die Dummy-Module für die Benutzer- Ein-/Ausgabe und die Benutzerausgänge. Der Modul flam_recu.o enthält keine Benutzer-Ein-/Ausgabe-Routinen und keine Benutzerausgänge.

Die Satzschnittstelle wird mit folgendem Kommando mit dem Anwenderprogramm (xyz) gebunden:

```
ld –o xyz /usr/lib/flame/flam_rec.o xyz.c –lc
```

Will der Anwender Benutzerausgänge und/oder eigene Ein-/Ausgabe-Routinen verwenden, müssen diese zum Anwenderprogramm gebunden werden. Hierbei ist statt flam_rec.o der Modul flam_recu.o zu verwenden.

Hierbei ist zu beachten, dass nicht nur einzelne Routinen vom Anwender zu erstellen sind, sondern alle Benutzerausgänge und/oder Ein-/Ausgabe-Routinen. Will der Anwender nur eigene Benutzerausgänge oder nur eigene Ein-/Ausgabe-Routinen verwenden, kann er für die übrigen Funktionen, die in der /usr/lib/flame vorhandenen Dummy-Routinen flam_usrmod.o bzw. flam_exitmod.o einbinden.

Das Kommando lautet dann:

```
ld –o xyz /usr/lib/flame/flam_recu.o
/usr/lib/flame/flam_usrmod.o
/usr/lib/flame/flam_exitmod.o xyz.c –lc
```

/usr/lib/flame/flam_usrmod.o und/oder

/usr/lib/flame/flam_exitmod.o sind durch Module des Anwenders zu ersetzen.

Bei einigen UNIX-Systemen muss die Runtime-Library /lib/crt0.o im Link-Kommando mit angegeben werden.

FLAM (UNIX)

Benutzerhandbuch

Kapitel 5:

Die Benutzer-Ein-/ Ausgabe-Schnittstelle

5. Die Benutzer-Ein-/Ausgabe-Schnittstelle

5.1 Anwendung der Benutzer-Ein-/ Ausgabe-Schnittstelle

FLAM benutzt für die Ein- und Ausgabe von Originaldateien, dekomprimierten Dateien und FLAMFILEs, die gewöhnlichen Funktionen. Es bietet jedoch auch die Möglichkeit, in Programmen, die die Unterprogrammschnittstelle oder die FLAM-Funktionen der Satzschnittstelle aufrufen, Daten in einem nicht unterstützten Format oder auf einem von UNIX nicht unter- stützten Gerät zu verarbeiten. Dies ist möglich, wenn die erforderlichen Ein- und Ausgabe-Routinen zur Verfügung gestellt werden. FLAM setzt für die betreffende Datei diese Routinen ein, wenn beim Aufruf der Unterprogrammschnittstelle der Parameter inuser_io für die Originaldatei, user_io für die FLAMFILE, outuser_io für die dekomprimierte Datei bzw. beim Aufruf der FLAM-Funktion flmopd im Argument device Benutzer-Ein-/Ausgabe spezifiziert wird (device=7).

Der wesentliche Unterschied zwischen der Benutzer-Ein-/Ausgabe-Schnittstelle und der Satzschnittstelle besteht darin, dass bei den Funktionen der Satzschnittstelle das Anwenderprogramm der aufrufende, FLAM der ausführende Teil ist. Benutzer-Ein-/Ausgabe hingegen sind selbst die ausführenden Routinen und werden von FLAM aufgerufen. Diese haben also keinen Einfluss auf die Reihenfolge, in der sie aktiviert werden und müssen bei jedem Aufruf situationsgerecht reagieren. Sofern sie hierfür Arbeitsspeicher benötigen, wird für jeden usropn-Aufruf der Datei ein Bereich von 1 KByte zugeordnet und mit jedem Folgeaufruf für diese Datei unverändert übergeben.

Da FLAM konzeptionell immer Daten aus einer Datei liest und in eine Datei schreibt, sind Benutzer-Ein-/Ausgabe-Routinen aus der Sicht von FLAM einfach alternative Dateizugriffe, unabhängig davon, welche realen Aktionen von ihnen ausgeführt werden.

Benutzer-Ein-/Ausgabe-Routinen müssen daher folgende Funktionen enthalten:

- usrcls
- usrget
- usropn
- usrpos
- usrput

Diese Funktionen müssen den Schnittstellenkonventionen genügen, d.h. die Funktionen müssen mit dem vorgegebenen Namen definiert sein, die Argumente in Typ und Länge übereinstimmen und die Returncodes den Erfolg oder die Fehlersituation korrekt widerspiegeln.

Hinweise für das Einbinden von Benutzer-Ein-/Ausgabe-Routinen finden sich am Ende dieses Abschnitts.

usrcls

Mit dieser Funktion wird eine Datei geschlossen.

Syntax

void **usrcls** (char ***workio*,
long **returncode*);

Argumente

workio

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf einer Benutzer-Ein-/Ausgabe-Funktion als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

usrget

Mit dieser Funktion wird ein Satz sequentiell gelesen und an FLAM übergeben.

Syntax

void **usrget** (char ***workio*,
long **returncode*,
long **record_length*,
char **record*,
long **buffer_length*);

Argumente

workio
Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf einer Benutzer-Ein-/Ausgabe-Funktion als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird.

returncode
In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

record_length
In diesem Argument ist die Länge des gelesenen Satzes in Bytes zurückzugeben.

record
Dieses Argument ist der Satzpuffer.

buffer_length
Dieses Argument enthält die Länge des Satzpuffers in Bytes.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

usropn

Mit dieser Funktion wird die Datei geöffnet, deren Spezifikation im Argument filename übergeben wird.

Syntax

void **usropn** (char ***workio*,
long **returncode*,
long **file_open*,
char **filename*,
long **organization*,
long **record_format*,
long **record_size*,
long **block_size*,
struct *kd *key_description*,
long **device*,
char **record_delimiter*,
char **pad_char*,
long **print_control*,
long **close_disposition*,
long **access*,
long **filename_length*,
char **filename*);

Argumente

workio

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf einer Benutzer-Ein-/Ausgabe-Funktion als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird.

Beim Aufruf von usropn steht am Anfang des Bereichs der Name der zu bearbeitenden Datei, der Rest ist mit Binärnullen vorbesetzt.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

file_open

Dieses Argument gibt den Verarbeitungsmodus der zu öffnenden Datei an.

0 input (lesen, d.h. dekomprimieren)
1 output (schreiben, d.h. komprimieren)

filename

In diesem Argument wird die Spezifikation für die zu öffnende Datei übergeben. Diese Zeichenkette ist mit einem Null-Byte (Binärnull) abgeschlossen. Diese Spezifikation kann auch ein logischer Name sein. Der tatsächliche Dateiname soll in diesem Argument zurückgegeben werden und die Namenslänge im Argument filename_length.

Der Inhalt dieses Arguments steht auch am Anfang des mit dem Argument workarea übergebenen Speicherbereichs.

organization
Dieses Argument ist für künftige Erweiterungen reserviert.

record_format
In diesem Argument wird ein Code für das Satzformat der zu öffnenden Datei übergeben bzw. zurückgegeben.
Die Werte sind wie folgt definiert:

0	variabel/var_2b
1	fix
2	undefined
3	stream
18	eaf
32	var_4b
40	var_ascii
48	var_ebcdic

record_size
In diesem Argument wird die Satzlänge der Datei übergeben bzw. kann für eine Eingabedatei zurückgegeben werden. Der Wert 0 wird bei variablen Satzformaten gesetzt. Ein Wert ungleich 0 ist bei den Satzformaten fix und undefined erforderlich.

block_size
Dieses Argument ist für künftige Erweiterungen reserviert.

key_description
Beim Öffnen einer indexsequentiellen Datei gibt dieses Argument den Speicherbereich an, in dem die Beschreibung des Schlüssels übergeben bzw. zurückgegeben wird.

Dieses Argument ist für künftige Erweiterungen reserviert.

device
Dieses Argument ist für künftige Erweiterungen reserviert.

record_delimiter
Beim Öffnen von Dateien mit Satzformat stream gibt dieses Argument an, welche Satztrennzeichen zu verwenden sind. Enthält das Argument den Wert 0, so gelten die Standard-Endezeichen. Sonst ist der Inhalt dieses Arguments das Satztrennzeichen.

Der record_delimiter hat folgenden Aufbau:

Im höchstwertigen Byte steht die Anzahl der Bytes. Bei Anzahl der Bytes=1 steht im niedrigstwertigen Byte das Satztrennzeichen. Bei Anzahl Bytes=2 steht im zweitniedrigstwertigen Byte das 1., im niedrigstwertigen Byte das 2. Byte des Satztrennzeichens, z.B.:

"0a 00 00 01"	Satztrennzeichen UNIX
"0a 0d 00 02"	Satztrennzeichen MS-DOS

pad_char
In diesem Argument wird ein Füllzeichen übergeben, das bei Ausgabe mit fixer Satzlänge zum Auffüllen kürzerer Sätze benutzt wird.

print_control Dieses Argument ist für künftige Erweiterungen reserviert.

close_disposition Dieses Argument ist für künftige Erweiterungen reserviert.

access Dieses Argument ist für künftige Erweiterungen reserviert.

filename_length In diesem Argument wird die Länge des Bereiches, in dem der Name (filename) der zu öffnenden Datei steht, übergeben und die Länge des Namens zurückgegeben.

filename Dieses Argument ist der Bereich, in dem der Name der zu öffnenden Datei übergeben bzw. zurückgegeben wird.

Returncodes Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

usrpos

Mit usrpos wird bei der Dekomprimierung einer relativen Datei der Satzschlüssel erhöht, wenn in der dekomprimierten Datei Lücken zu erzeugen sind.

Diese Funktion wird nicht benutzt und ist deshalb leer.

Syntax

void **usrpos** (char ***workio*,
long **returncode*,
long **position*);

Argumente

workio

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf einer Benutzer-Ein-/Ausgabe-Funktion als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird.

returncode

In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

position

In diesem Argument wird die Anzahl zu überspringender Satzpositionen übergeben.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

usrput

Mit dieser Funktion wird ein von FLAM übergebener Satz sequentiell geschrieben.

Syntax

void **usrput** (char ***workio*,
long **returncode*,
long **record_length*,
char **record*);

Argumente

workio — Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf einer Benutzer-Ein-/Ausgabe-Funktion als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird.

returncode — In diesem Argument wird ein Returncode an das aufrufende Programm zurückgegeben.

record_length — Dieses Argument enthält die Satzlänge in Bytes.

record — Dieses Argument ist der Satzpuffer.

Returncodes

Eine Beschreibung der Returncodes befindet sich in Anhang A, Returncodes.

5.2 Einbinden von Benutzer-Ein-/Ausgabe-Routinen in Anwenderprogramme

Der Datenzugriff über Benutzer-Ein-/Ausgabe-Routinen wird von FLAM bei Aufrufen sowohl der Unterprogramm- als auch der Satzschnittstelle unterstützt.

Um mit FLAM über Benutzer-Ein-/Ausgabe-Routinen auf die Daten zugreifen zu können, müssen die eingebundenen zugehörigen Objektdateien für alle oben genannten Zugriffsfunktionen vorhanden sein; nicht benötigte Funktionen können leer sein.

Das Binden der Routinen mit dem Anwenderprogramm erfolgt gemäß der in Abschnitt 3.3, "Binden von flamup " beschriebenen Vorgehensweise, bzw. wie in in Abschnitt 4.5 "Einbinden von Funktionen der Satzschnittstelle in Anwenderprogramme" beschrieben.

FLAM (UNIX)

Benutzerhandbuch

Kapitel 6:

Die Benutzerausgänge

6. Die Benutzerausgänge

Ein Benutzerausgang ist ein Programm, das vom Benutzer zur Verfügung gestellt und von FLAM aktiviert wird und mit diesem über eine vereinbarte Schnittstelle kommuniziert.

FLAM unterstützt zwei Arten von Benutzerausgängen:

- Ausgänge, die bei Zugriffen auf Dateien, d.h. beim Öffnen, Schließen, Lesen oder Schreiben aktiviert werden, kurz auch Zugriffsexits genannt;

- einen Ausgang für eine automatische Schlüsselverwaltung, kurz Schlüsselexit.

Für das Zusammenwirken mit FLAM müssen die Zugriffsexits mit FLAM (statisch) gebunden werden (s. Abschnitt 6.1.2). Der Schlüsselexit hingegen wird als "Shared Object" erzeugt und wird dynamisch, d.h. zum Ausführungszeitpunkt von FLAM eingebunden. Näheres hierzu s. Abschnitt 6.2.2.

6.1 Benutzerausgänge für Dateizugriffe (Zugriffsexits)

Diese Benutzerausgänge sind Programme, die von FLAM mit jedem Zugriff auf die assoziierte Datei aktiviert werden. Beim Öffnen der Datei und bei Lesezugriffen wird der Benutzerausgang sofort nach Öffnen bzw. nach der Übertragung eines Satzes in den Lesepuffer aktiviert, bevor die Verarbeitung durch FLAM beginnt.

Beim Schließen und bei Schreibzugriffen wird er nach Abschluss der Verarbeitung durch FLAM und unmittelbar vor dem Schließen bzw. vor der Ausgabe aktiviert. Der Benutzerausgang kann den betreffenden Satz auswerten oder manipulieren oder auch Sätze einfügen und gibt anschließend die Kontrolle zurück an FLAM. Zugleich erwartet FLAM vom Benutzerausgang über einen Returncode Anweisungen, wie die Verarbeitung fortzusetzen bzw. ob sie evtl. aufgrund einer Fehlersituation abzubrechen ist.

Über das Unterprogramm flamup können Zugriffsexits sowohl für FLAMFILEs als auch für Original- bzw. dekomprimierte Dateien aktiviert werden. Über die Satzschnittstellenfunktionen können nur Ausgänge für Zugriffe auf die FLAMFILE aktiviert werden, da die Original- und die dekomprimierten Dateien dann nicht von FLAM gelesen oder beschrieben werden.

FLAM besitzt folgende Zugriffsexits für die Komprimierung:

exk10 Dieser wird nach jedem Lesen eines Satzes der Originaldatei aktiviert.

exk20 Dieser wird vor jedem Schreiben eines Satzes der FLAMFILE aktiviert.

FLAM besitzt folgende Zugriffsexits für die Dekomprimierung:

exd10 Dieser wird vor jedem Schreiben eines Satzes der dekomprimierten Datei aktiviert.

exd20 Dieser wird nach jedem Lesen eines Satzes der FLAMFILE aktiviert.

Die Benutzerausgänge müssen mit der Unterprogrammschnittstelle (flam_upu.o) bzw. der Satzschnittstelle (flam_recu.o) und dem Anwenderprogramm gebunden werden. Dadurch kann nur eine Funktion je Benutzerausgang definiert werden. In einem Anwenderprogramm nicht benötigte Benutzerausgänge müssen als leere Funktion vorhanden sein. (Binden der Programme, siehe Kapitel 3.3 bzw. 4.5)

Beispiele für Benutzerausgänge sowie ein Programm zum Komprimieren und Dekomprimieren von Dateien (analog FLAM) sind in Kapitel 7 enthalten.

6.1.1 Programmierung der Zugriffsexits

Bei der Programmierung der Benutzerausgänge für Dateizugriffe ist das Umfeld zu berücksichtigen, in dem diese ausgeführt werden. Beispielsweise dürfen sie keinerlei interaktive Benutzereingriffe wie Tastatureingaben erfordern, wenn ihre Verwendung im Stapelbetrieb nicht auszuschließen ist. Auch können Bildschirmausgaben das von FLAM erzeugte Protokoll in unerwünschter Weise verändern.

Die mittels Pointer zur Verfügung gestellten Speicherbereiche dürfen nur in der im Aufruf angegebenen Länge modifiziert werden. Speicher für einzufügende Sätze ist vom Exit selbst anzufordern und freizugeben.

Im Folgenden ist die Syntax der Schnittstellen zwischen FLAM und den Benutzerausgängen im Detail beschrieben.

exd10

Dieser Benutzerausgang kann bei der Dekomprimierung für die Zugriffe auf die dekomprimierte Datei zur Nachbearbeitung der auszugebenden Sätze benutzt werden. Er wird durch das Unterprogramm flamup über den Parameter exd10=*Exitspezifikation* im Argument parameter_string aktiviert.

Er erhält die Kontrolle nach dem Öffnen und vor jedem Schreibzugriff und dem Schließen der dekomprimierten Datei, wobei ihm vor Schreibzugriffen der zu schreibende Satz zur Verfügung gestellt wird. Bei Rückgabe der Kontrolle an FLAM übergibt er einen Returncode.

Syntax

void **exd10** (long* *functioncode*,
long* *returncode*,
char** *record_pointer*,
long* *record_length*,
char* *work*)

Argumente

functioncode

In diesem Argument wird dem Benutzerausgang ein Funktionscode übergeben. Gültige Werte sind:

0	Erster Aufruf für die Datei nach open
4	Satz im Satzpuffer bereitgestellt
8	Letzter Aufruf für die Datei vor close

returncode

In diesem Argument gibt der Benutzerausgang einen Returncode an FLAM zurück (Siehe besondere Beschreibung der Returncodes und ihrer Bedeutungen).

record_pointer

Dieses Argument enthält einen Pointer auf einen Character String. Bei Aufrufen mit Funktionscode 4 enthält er die Adresse des zu schreibenden Satzes. Diese Adresse kann vom Benutzerausgang geändert werden, beispielsweise um einen Satz einzufügen.

record_length

Dieses Argument enthält bei Aufrufen mit Funktionscode 4 die Länge des zu schreibenden Satzes. Diese Länge kann vom Benutzerausgang geändert werden, beispielsweise beim Einfügen eines Satzes.

work

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf des Benutzerausgangs als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird. Bei erstmaligem Aufruf (Funktionscode 0) enthält der Bereichsanfang den Dateinamen, der Rest ist mit Binärnullen initialisiert.

Returncodes	Wert	Bedeutung
	0	Satz übernehmen bzw. kein Fehler
	4	Satz nicht übernehmen
	8	Satz einfügen. Die Pufferadresse des einzufügenden Satzes und seine Länge müssen in den Argumenten record_pointer und record_length zurückgegeben werden. Der Benutzerausgang wird erneut für den ursprünglichen Satz aufgerufen.
	12	Ende der Komprimierung einleiten
	16	Fehler im Benutzerausgang; abnormales Ende

exd20

Dieser Benutzerausgang kann bei der Dekomprimierung für die Zugriffe auf die FLAMFILE zur Vorverarbeitung der gelesenen Sätze benutzt werden. Er kann durch das Unterprogramm flamup oder die Satzschnittstellenfunktion flmopf aktiviert werden.

Im Unterprogrammaufruf wird er durch den Parameter exd20=*Exitspezifikation* im Argument parameter_string angegeben. Bei der Funktion flmopf wird er im Argument exitroutine_decomp spezifiziert.

Er erhält die Kontrolle nach dem Öffnen, nach jedem Lesezugriff und vor dem Schließen der FLAMFILE, wobei ihm nach Lesezugriffen der gelesene Satz zur Verfügung gestellt wird. Bei Rückgabe der Kontrolle an FLAM übergibt er einen Returncode.

Syntax

void **exd20** (long **functioncode*,
long **returncode*,
char ***record_pointer*,
long **record_length*,
char **work*)

Argumente

functioncode

In diesem Argument wird dem Benutzerausgang ein Funktionscode übergeben. Gültige Werte sind:

0	Erster Aufruf für die Datei nach open
4	Satz im Satzpuffer bereitgestellt
8	Letzter Aufruf für die Datei vor close

returncode

In diesem Argument gibt der Benutzerausgang einen Returncode an FLAM zurück (Siehe besondere Beschreibung der Returncodes und ihrer Bedeutungen).

record_pointer

Dieses Argument enthält einen Pointer auf einen Character String. Der Pointer hat eine Länge von 4 Byte und entspricht damit einem long integer. Bei Aufrufen mit Funktionscode 4 enthält er die Adresse des gelesenen Satzes. Diese Adresse kann vom Benutzerausgang geändert werden, beispielsweise um einen Satz einzufügen.

record_length

Dieses Argument enthält bei Aufrufen mit Funktions- code 4 die Länge des gelesenen Satzes. Diese Länge kann vom Benutzerausgang geändert werden, beispielsweise beim Einfügen eines Satzes.

work — Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf des Benutzerausgangs als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird. Bei erstmaligem Aufruf (Funktionscode 0) enthält der Bereichsanfang den Dateinamen, der Rest ist mit Binärnullen initialisiert.

Returncodes

Wert	Bedeutung
0	Satz übernehmen bzw. kein Fehler.
4	Satz nicht übernehmen.
8	Satz einfügen. Die Pufferadresse des einzufügenden Satzes und seine Länge müssen in den Argumenten record_pointer und record_length zurückgegeben werden. Der Benutzerausgang wird erneut für den ursprünglichen Satz aufgerufen.
12	Ende der Komprimierung einleiten.
16	Fehler im Benutzerausgang; abnormales Ende.

exk10

Dieser Benutzerausgang kann bei der Komprimierung für die Bearbeitung der Sätze der Originaldatei nach dem Lesen benutzt werden. Er wird durch das Unterprogramm flamup über den Parameter exk10=*Exitspezifikation* im Argument parameter_string aktiviert.

Er erhält die Kontrolle nach dem Öffnen, nach jedem Lesezugriff und vor dem Schließen der Originaldatei, wobei ihm nach Lesezugriffen der gelesene Satz zur Verfügung gestellt wird. Bei Rückgabe der Kontrolle an FLAM übergibt er einen Returncode.

Syntax

void **exk10** (long **functioncode*,
long **returncode*,
char ***record_pointer*,
long **record_length*,
char **work*)

Argumente

functioncode

In diesem Argument wird dem Benutzerausgang ein Funktionscode übergeben. Gültige Werte sind:

0	Erster Aufruf für die Datei nach open
4	Satz im Satzpuffer bereitgestellt
8	Letzter Aufruf für die Datei vor close

returncode

In diesem Argument gibt der Benutzerausgang einen Returncode an FLAM zurück (Siehe besondere Beschreibung der Returncodes und ihrer Bedeutungen).

record_pointer

Dieses Argument enthält einen Pointer auf einen Character String. Der Pointer hat eine Länge von 4 Byte und entspricht damit einem long integer. Bei Aufrufen mit Funktionscode 4 enthält er die Adresse des gelesenen Satzes. Diese Adresse kann vom Benutzerausgang geändert werden, beispielsweise um einen Satz einzufügen.

record_length

Dieses Argument enthält bei Aufrufen mit Funktions- code 4 die Länge des gelesenen Satzes. Diese Länge kann vom Benutzerausgang geändert werden, beispielsweise beim Einfügen eines Satzes.

work

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf des Benutzerausgangs als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird. Bei erstmaligem Aufruf (Funktionscode 0) enthält der Bereichsanfang den Dateinamen, der Rest ist mit Binärnullen initialisiert.

Returncodes	Wert	Bedeutung
	0	Satz übernehmen bzw. kein Fehler
	4	Satz nicht übernehmen
	8	Satz einfügen. Die Pufferadresse des einzufügenden Satzes und seine Länge müssen in den Argumenten record_pointer und record_length zurückgegeben werden. Der Benutzerausgang wird erneut für den ursprünglichen Satz aufgerufen.
	12	Ende der Komprimierung einleiten
	16	Fehler im Benutzerausgang; abnormales Ende

exk20

Dieser Benutzerausgang kann bei der Komprimierung für die Bearbeitung der Sätze der FLAMFILE vor dem Schreiben benutzt werden. Er kann durch das Unterprogramm flamup oder die Satzschnittstelle aktiviert werden.

Im Unterprogrammaufruf wird er durch den Parameter exk20=*Exitspezifikation* im Argument parameter_string angegeben. Bei der Funktion flmopf wird er im Argument exitroutine_comp spezifiziert.

Er erhält die Kontrolle nach dem Öffnen und vor jedem Schreibzugriff und dem Schließen der FLAMFILE, wobei ihm vor Schreibzugriffen der zu schreibende Satz zur Verfügung gestellt wird. Bei Rückgabe der Kontrolle an FLAM übergibt er einen Returncode.

Syntax

void **exk20** (long **functioncode*,
long **returncode*,
char ***record_pointer*,
long **record_length*,
char **work*);

Argumente

functioncode

In diesem Argument wird dem Benutzerausgang ein Funktionscode übergeben. Gültige Werte sind:

0	Erster Aufruf für die Datei nach open
4	Satz im Satzpuffer bereitgestellt
8	Letzter Aufruf für die Datei vor close

returncode

In diesem Argument gibt der Benutzerausgang einen Returncode an FLAM zurück (Siehe besondere Beschreibung der Returncodes und ihrer Bedeutungen).

record_pointer

Dieses Argument enthält einen Pointer auf einen Character String. Der Pointer hat eine Länge von 4 Byte und entspricht damit einem long integer. Bei Aufrufen mit Funktionscode 4 enthält er die Adresse des zu schreibenden Satzes der FLAMFILE.

record_length

Dieses Argument enthält bei Aufrufen mit Funktionscode 4 die Länge des zuschreibenden Satzes.

work

Dieses Argument ist ein Bereich von 1 KByte, der von FLAM bei jedem Aufruf des Benutzerausgangs als Arbeitsbereich zur Verfügung gestellt wird und zwischen den Aufrufen von FLAM nicht verändert wird. Bei erstmaligem Aufruf (Funktionscode 0) enthält der Bereichsanfang den Dateinamen, der Rest ist mit Binärnullen initialisiert.

Returncodes	Wert	Bedeutung
	0	Satz übernehmen bzw. kein Fehler
	4	Satz nicht übernehmen
	8	Satz einfügen. Die Pufferadresse des einzufügenden Satzes und seine Länge müssen in den Argumenten record_pointer und record_length zurückgegeben werden. Der Benutzerausgang wird erneut für den ursprünglichen Satz aufgerufen. Bei der Dekomprimierung müssen so eingefügte Sätze durch einen komplementären Benutzerausgang wieder entfernt werden.
	12	Ende der Komprimierung einleiten
	16	Fehler im Benutzerausgang; abnormales Ende

6.1.2 Einbinden der Zugriffsexits in Anwenderprogramme

Der Aufruf von Benutzerausgängen für Dateizugriffe wird in FLAM sowohl von der Unterprogramm- als auch der Satzschnittstelle unterstützt, wobei dies bei letzterer lediglich in Bezug auf FLAMFILE-Zugriffe gilt.

Um mit FLAM über Benutzerausgänge Datensätze bearbeiten zu können, müssen die eingebundenen zugehörigen Objektdateien alle oben genannten Zugriffsfunktionen enthalten; nicht benötigte Funktionen können leer sein.

Die Benutzerausgänge für Dateizugriffe werden mit FLAM statisch gebunden gemäß der in Abschnitt 3.3, "Binden von flamup " beschriebenen Vorgehensweise, bzw. wie in in Abschnitt 4.5 "Einbinden von Funktionen der Satzschnittstelle in Anwenderprogramme" beschrieben.

6.2 Der Benutzerausgang für automatische Schlüsselverwaltung (Schlüsselexit)

Der Benutzerausgang für automatische Schlüsselverwaltung ist über die Unterprogrammschnittstelle aktivierbar und - im Gegensatz zu den Zugriffsexits - auch über das flam-Kommando, nicht jedoch über die Satzschnittstelle.

Ein weiterer Unterschied zu den Zugriffsexits ist, dass die Schlüsselexit-Routine dynamisch gebunden wird. Sie wird als shared object erzeugt, und sowohl der Name der shared library wie der der aufgerufenen Funktion sind frei wählbar und werden FLAM als Parameter übergeben.

6.2.1 Programmierung des Schlüsselexits

Auch Schlüsselexits dürfen keine Benutzer-Interaktionen enthalten, falls sie für die Stapelverarbeitung geeignet sein sollen. Konsolausgaben können mit der FLAM-Protokollierung synchronisiert werden. Dazu dienen die Argumente *message* und *msglen* sowie der Funktionscode -1 (Versionsidentifikation). FLAM ruft bei jeder Ausführung den Exit einmal mit diesem Funktionscode auf um einen ggf. zurückgereichten Meldungstext an geeigneter Stelle im Protokoll auszugeben. Bei den anderen Funktioncodes sollten Meldungstexte nur in Fehlerfällen zurückgegeben werden mit einem Hinweis auf den Abbruchgrund.

Im Folgenden ist die Syntax der Schnittstelle zwischen FLAM und dem Schküsselexit detailliert beschrieben.

kmfunc

Dieser Benutzerausgang kann bei der Ver- und Entschlüsselung benutzt werden, um FLAM automatisch ein Kennwort zur Verfügung zu stellen. Er wird über den Parameter -kmexit=*Exitangaben* im flam-Kommando oder im Argument parameter_string des Unterprogramms flamup aktiviert. Der tatsächliche, anstelle von *kmfunc* eingesetzte Funktionsname kann frei gewählt werden.

Bei Rückgabe der Kontrolle an FLAM übergibt er einen Returncode.

Syntax

void ***kmfunc*** (signed long **functioncode*,
signed long **returncode*,
const unsigned long **parmlen*,
const unsigned char **param*,
unsigned long **datalen*,
unsigned char **data*,
unsigned long **ckylen*,
unsigned char **cryptokey*,
unsigned long **msglen*,
unsigned char **message*);

Argumente

functioncode

In diesem Argument wird dem Benutzerausgang ein Funktionscode übergeben.

Gültige Werte sind:

-1	Aufruf zur Versionsidentifikation
0	Aufruf zur Entschlüsselung
1	Aufruf zur Verschlüsselung

returncode

In diesem Argument gibt der Benutzerausgang einen Returncode an FLAM zurück (Siehe besondere Beschreibung der Returncodes und ihrer Bedeutungen).

parmlen

Dieses Argument enthält bei Aufrufen mit Funktionscodes 0 oder 1 die Bytelänge der Daten im Argument *param*. Diese kann zwischen 0 und 256 liegen.

param

Falls *parmlen* > 0, steht hier der *exparm*-Teil der *Exitangaben* des Parameters -kmexit (s. Beschreibung des Parameters -kmexit im Abschnitt 2.3).

datalen

Dieses Argument enthält bei Aufrufen mit Funktionscodes 0 oder 1 die Bytelänge der Daten im Argument *data*. Diese kann zwischen 0 und 512 liegen. Bei Aufrufen zur Verschlüsselung wird sie vom Benutzerausgang, Bei Aufrufen zur Entschlüsselung von FLAM gesetzt.

data — Falls *datalen* > 0, liefert bei Aufrufen zur Verschlüsselung (Funktionscode 1) der Benutzerausgang hier die Daten, die er bei Aufrufen zur Entschlüsselung zum Auffinden des Schlüssels braucht. FLAM speichert diese in einem anwenderspezifischen FLAM-Fileheader und stellt sie bei Aufrufen zur Entschlüsselung (Funktionscode 0) dem Benutzerausgang hier zu Verfügung.

ckylen — In diesem Argument setzt der Benutzerausgang bei erfolgreichen Aufrufen mit Funktionscodes 0 oder 1 die Bytelänge des an FLAM im Argument *cryptokey* übergebenen Schlüssels. Die maximale Schlüssellänge ist 64 Bytes.

cryptokey — Hier empfängt FLAM bei Funktionscodes 0 oder 1 vom Benutzerausgang den Schlüssel für die Ver- bzw. Entschlüsselung in der Länge aus dem Argument *ckylen*.

msglen — In dieses Feld stellt der Benutzerausgang die Bytelänge der Nachricht im Argument *message*. Maximale Länge ist 128 Bytes.

message — Hier stellt der Benutzerausgang eine Nachricht ein, die von FLAM ggf. im Protokoll ausgegeben wird.

Returncodes

Wert	Bedeutung
0	kein Fehler
sonst	Fehler

6.2.2 Erzeugung des Schlüsselexits

Der Name der Schlüsselexit-Funktion kann frei gewählt werden. Diese Funktion muss in einer Shared Library enthalten sein, die von FLAM während der Ausführung geöffnet wird. Informationen zur Erzeugung von Shared Libraries entnehmen Sie bitte der Dokumentation des verwendeten Compilers.

FLAM (UNIX)

Benutzerhandbuch

Kapitel 7:

Anwendungsbeispiele

7. Anwendungsbeispiele

In diesem Kapitel wird die Benutzung von FLAM anhand von Beispielen illustriert. Für das flam-Kommando veranschaulichen zahlreiche Beispiele die vorhandenen Möglichkeiten, während für die Programmschnittstellen C-Programme oder Ausschnitte aus solchen als Vorlage für den praktischen Gebrauch dienen. Diese Bei- spiele werden ggf. durch Hinweise für das Binden der Programme ergänzt.

7.1 Kommandos

In den Beispielen werden nur die erforderlichen Parameter benutzt. Sollen Parameter von den Standard- werten abweichende Werte haben, müssen diese angegeben werden.

7.2 Komprimieren

7.2.1 Eine Datei in eine Datei

Die Datei test.dat wird komprimiert und die komprimierten Daten werden in die Datei test.cmp geschrieben.

flam -compress -flamin=test.dat -flamfile=test. cmp

7.2.2 Mehrere Dateien in eine Datei

Die Dateien mit dem Suchmuster t*.dat werden komprimiert und die komprimierten Daten werden in die Datei test.cmp geschrieben.

flam -compress -flamin=t*.dat -flamfile=test.cmp -attributes=all -show=all

Mit -show=all werden alle Informationen zur Komprimierung angezeigt.

-attributes=all bewirkt, dass die Namen, die Datei- und die Satzattribute der Originaldateien in der Komprimatsdatei gespeichert werden. Diese können dann mit dem Kommando

flam -decompress -show=attributes -flamfile=test.cmp

angezeigt werden, ohne die dekomprimierten Dateien zu erzeugen.

7.2.3 Mehrere Dateien in jeweils eine Datei

Die Dateien mit dem Suchmuster t*.dat werden komprimiert. Die komprimierten Daten jeweils einer Datei werden in eine Datei mit dem Namen der Originaldatei und dem Suffix cmp geschrieben:

flam -compress -flamin= t*.dat -flamfile=[dat=cmp] \
-mode=adc

Enthielte das Default-Verzeichnis etwa auch eine Datei tvdaten.dat, so würde deren Komprimat in der FLAMFILE tvcmpen.dat gespeichert (s. Abschnitt 2.3.2, Ausgabespezifikationen).

Um sicherzugehen, dass die gewünschte Substitution tatsächlich auf das Suffix angewandt wird, ist die folgende Schreibweise vorzuziehen:

flam -compress -flamin=t*.dat -flamfile=[.dat=.cmp] \
-mode=adc

7.3 Dekomprimieren

7.3.1 Eine Datei in eine Datei

Die Datei test.cmp wird dekomprimiert und die dekomprimierten Daten werden in die Datei test.dat geschrieben. Die komprimierte Datei kann aus einer oder mehreren Originaldateien bestehen.

flam -decompress -flamfile=test.cmp -flamout=test.dat

7.3.2 Eine Komprimatsdatei, bestehend aus mehreren Originaldateien, in jeweils eine der Originaldatei gleiche Datei

Eine Komprimatsdatei, die die Daten mehrerer Originaldateien jeweils mit FLAM-Fileheader (-attributes=all) enthält, wird dekomprimiert.

Die dekomprimierten Daten jeweils einer Originaldatei werden in eine Datei gleichen Namens mit gleichen Attributen geschrieben:

flam -decompress -flamfile=test.cmp -flamout=[*]

7.3.3 Mehrere Dateien in eine Datei

Die Dateien mit dem Suchmuster t*.cmp werden dekomprimiert und die dekomprimierten Daten werden in die Datei test.dat geschrieben:

flam -decompress -flamfile=t*.cmp -flamout=test.dat

7.3.4 Mehrere Dateien in jeweils eine Datei

Die Dateien mit dem Suchmuster t*.cmp werden dekomprimiert. Die dekomprimierten Daten werden jeweils in eine Datei gleichen Namens mit dem Suffix dat geschrieben:

flam -decompress -flamfile=t*.cmp -flamout=[.cmp=.dat]

Hierbei ist zu beachten, dass sämtliche Komprimate aus einer einzelnen FLAMFILE in eine einzige Datei dekomprimiert werden.

Wurde mit -attributes=all komprimiert, kann mit

flam -decompress -flamfile=t*.cmp -flamout=[*]

dekomprimiert werden, um die dekomprimierten Dateien mit den Originalnamen und -attributen zu erzeugen.

7.4 Verwendung einer Parameterdatei

Die Parameter für FLAM können nicht nur über das Kommando angegeben werden. Sie können auch in einer Parameterdatei stehen, die im Kommando angegeben wird:

flam -parfile=param.dat

Die Datei param.dat enthält die Parameter:

Für die Komprimierung:

flamin=test.dat

flamfile=test.cmp

comp

Für die Dekomprimierung:

flamfile=test.cmp

flamout=test.dat

decomp

7.5 Verwendung der Unterprogramm-schnittstelle

Dateien können in einem Anwenderprogramm über die Unterprogrammschnittstelle flamup komprimiert und dekomprimiert werden. Der Aufruf in einem C-Anwenderprogramm lautet:

flamup(id, rc, par_string, parlen);

id ist die Adresse eines 4 Byte langen Feldes.

rc ist die Adresse eines 4 Byte langen numerischen Feldes, das nach Ablauf von flamup den Returncode enthält.

par_string ist die Adresse eines Strings, der die Parameter enthält.

parlen ist die Adresse eines 4 Byte langen numerischen Feldes, das die Länge von par_string enthält.

par_string enthält z.B.:

flamin=test.dat,flamfile=test.cmp,comp

parlen enthält dann den Wert 38.

Das folgende Beispielprogramm verwendet die Unterprogrammschnittstelle. Die Parameter werden im Aufruf angegeben analog dem Kommando flam.

Dieses Programm kann als Anwenderprogramm mit Benutzer- Ein-/Ausgabe-Routinen und/oder Benutzerausgängen verwendet werden:

```
/*

Aufruf von flamup mit user-exit und/oder user-i/o

*/

#include <<stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "flamincl.h"

void main (argc, argv)
int argc;                       /* Anzahl Parameter in Kommando */
char *argv[];                   /* uebergebene Parameter */

{
char *id;                       /* Kennung */

unsigned long retco;            /* Return Code */
long ip, is;                    /* integer */
char *po, *pq;                        /* Pointer */
unsigned char *pp;                    /* Pointer */

char string[100];                     /* Characterstring */
char parstring[1500];                 /* Parameterstring */

retco = FLAM_NORMAL;                  /* Return Code = ok */

pp = parstring;

is = 0;
ip = 0;
while (++ip < argc)
  { po = argv[ip];                    /* Parameter nach Eingabebereich */
    pq = &string[0];
    while (*po != 0x00)
      { if ((*po != '-') && (*po != '+'))
          *pq++ = *po++;
          else
              po++;
     };
   *pq = 0x00;                       /* enthaelt Parameter Listen ? */
   po = &string[0];
   while ((*po != 0x00) && (*po != ',') && (*po != ' '))
     po++;
```

```
   if (*po == 0x00)                     /* Parameter ohne Listen */
     { po = &string[0];                 /* nach Parameterstring */
       while (*po != 0x00)
         { *pp++ = *po++;
           is++;
         };
    }
    else
        { po = &string[0];              /* Parameter mit Liste */
          do                            /* "d1 d2 ..." */
            { *pp++ = *po;              /* "d1,d2,..."     */
              is++;                     /* d1,d2,... */
            } while (*po++ != '=');     /* nach Parameterstring */
          *pp++ = '(';
          is++;
          if (*po == '"')
            po++;
          while (*po != 0x00)
            { if ((*po != ',') && (*po != '"') && (*po != ' ') )
                { *pp++ = *po++;
                  is++;
                }
                else
                     { if (*po == '"')
                         po++;
                         else
                              { *pp++ = ',';
                                po++;
                                is++;
                              }
                     };
           };
           if (*(pp - 1) == ',')
             { pp--;
               is--;
             };
           *pp++ = ')';
           is++;
         };
       *pp++ = ',';
       is++;
     };
if (is != 0)
  { *--pp = ' ';
    is--;
  };
```

```
/*

        Aufruf flamup

*/

flamup(&id, &retco, parstring, &is);

put_flmsg(retco);               /* Ausgabe des Returncodes */

exit(retco);

}
```

7.6 Verwendung der Satzschnittstelle

7.6.1 Komprimieren einer Datei

In einem Anwenderprogramm werden Datensätze gelesen und an die Satzschnittstelle von FLAM zur Komprimierung übergeben. Die Informationen über die Originaldatei und vom Anwender erzeugte Informationen werden im Fileheader gespeichert.

Die Namen der Ein- und Ausgabedatei werden im Dialog eingegeben.

```
/*

Beispiel: Komprimieren einer Datei
File Header mit allgemeinem und Anwenderteil wird geschrieben

*/

#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
extern int errno;
extern int sys_nerr;

#include "flamincl.h"

main  ()

{

char *flmid;                     /* Kennung Satzschnittstelle */
char kname[100];                 /* Name Komprimatsdatei */
char oname[100];                 /* Name Originaldatei */
long flcode;                     /* Flamcode */
long modus;                      /* Modus der Komprimierung */
long statis;                     /* Statistik */
long opmode;                     /* Openmodus */
long blkmode;                    /* Blockmodus */
long header;                     /* File Header */
long prctrl;                     /* Vorschubsteuerzeichen */
long reclength;                  /* Satzlaenge */
long namlen;                     /* Laenge Dateiname */
long sattrlen;                   /* Laenge systemspez. Information */
union { char c[4];
        char *p;
      } sysattr;                 /* systemspez. Information */
char datrec[2048];               /* Datenbereich */
char system[2];                  /* Betriebssystem */
```

```
long sanz;                      /* max. Satzanzahl / Block */
unsigned long maxb;             /* max. Puffergroesse in KB */
long lstpar;                    /* Letzter Parameter */
                                /* Komprimatsdatei */
long blksk;                     /* Blocksize */
long recfk;                     /* Satzformat */
long recsk;                     /* Satzlaenge */
long devk;                      /* Geraet */
long orgak;                     /* Organisation */
union { char c[4];
        unsigned long i;
      } recdelk;                /* Satztrenner */
long cldispk;                   /* Close Disposition */
struct kd keydesck;             /* Schluesselbeschreibung */
                                /* Originaldatei */
long devo;                      /* Geraet */
long orgao;                     /* Organisation */
long blkso;                     /* Blocksize */
long recfo;                     /* Satzformat */
long recso;                     /* Satzlaenge bei fix */
union { char c[4];
        unsigned long i;
      } recdelo;                /* Satztrenner */
long cldispo;                   /* Close Disposition */
struct kd keydesco;             /* Schluesselbeschreibung */

char exk20[34];                 /* Name Exitroutine */
char exd20[34];                 /* Name Exitroutine */

char *ptr;                      /* Pointer */
long il;                        /* Integer */
unsigned long rc;               /* Return Code */
unsigned long cputime;          /* CPU-Zeit */

unsigned long zks;              /* Satzzaehler */
unsigned long zkb;              /* Bytezaehler */
unsigned long zkbofl;           /* Bytezaehler Ueberlauf */
unsigned long zunks;            /* Satzzaehler */
unsigned long zunkb;            /* Bytezaehler */
unsigned long zunkbofl;         /* Bytezaehler Ueberlauf */

FILE *infile;                   /* Eingabedatei */

long uattrlen;                  /* Laenge Anwenderinformation */
static char usrattr[50] = {"Die Datei enthaelt Daten zur Anwendung xyz"};;

                                /* Dateinamen eingeben */
printf("Dateiname input: ");
scanf("%s", oname);
printf("Dateiname output: ");
scanf("%s", kname);
                                /* Open Originaldatei */
infile = fopen(oname, "r");
if (infile == NULL)
   exit(1);
```

```
                                  /* open FLAM */
opmode = FLAM_C_OPEN_OUTPUT;      /* komprimieren */
statis = FLAM_C_STATISTIC;        /* Statistik */
lstpar = FLAM_C_MORE_PARAMETER;
flmopn(&flmid, &rc, &lstpar, &opmode, kname, &statis);
if (rc != FLAM_NORMAL)
  closinput(rc, infile);
                                  /* Angaben zur Komprimatsdatei */
orgak = FLAM_C_ORG_SEQ;
recfk = FLAM_C_RECFRM_FIX;
recsk = 512;
blksk = 0;
cldispk = FLAM_C_CLOSE_REWIND;
devk = FLAM_C_DEVICE_DISK;
keydesck.keyparts = 0;
il = 100;
flmopd(&flmid, &rc, &lstpar, &il, kname, &orgak, &recfk,  &recsk,
       recdelk.c, &keydesck, &blksk, &cldispk, &devk);
if (rc != FLAM_NORMAL)
  closinput(rc, infile);
                                  /* Angaben zur Komprimierung */
il = FLAM_C_VERSION;              /* Version 2 */
flcode = FLAM_C_CHAR_ASCII;       /* Code der Komprimatsdatei */
modus = FLAM_C_MODUS_CX8;         /* Modus EIGHT_BIT */
maxb = 32768;                     /* Puffergroesse */
sanz = 255;                       /* Anzahl Saetze pro Block */
header = FLAM_C_FILEHEADER;       /* File Header */
blkmode = FLAM_C_BLOCKMODE;       /* Blockung FLAMFILE */
keydesco.keyparts = 0;
exk20[0] = ' ';                   /* keine Exitroutine */
exd20[0] = ' ';
flmopf(&flmid, &rc, &il, &flcode, &modus, &maxb, &header,
       &sanz, &keydesco, &blkmode, exk20, exd20);
if (rc != FLAM_NORMAL)
  closinput(rc, infile);

                                  /* File Header ausgeben */
namlen = strlen(oname);
orgao = FLAM_C_ORG_SEQ;           /* Organisation seq. */
recfo = FLAM_C_RECFRM_STREAM;     /* Satzformat stream, Textdatei */
recso = 0;
blkso = 0;
prctrl = FLAM_C_PRINT_CTRL_NONE;
system[0] = FLAM_C_SYSTEM_COMP; /* Computer / Prozessor */
system[1] = FLAM_C_SYSTEM_OS;   /* Betriebssystem */
flmphd(&flmid, &rc, &namlen, oname, &orgao, &recfo, &recso,
       recdelo.c, &keydesco, &blkso, &prctrl, system, &lstpar);
if (rc != FLAM_NORMAL)
  closfiles(rc, infile, &flmid);
                                  /* anwenderspez. File Header ausgeben */
```

```
uattrlen = strlen(usrattr);
flmpuh(&flmid, &rc, &uattrlen, usrattr);
if (rc != FLAM_NORMAL)
  closfiles(rc, infile, &flmid);

                                 /* Datei lesen und komprimieren */
il = FLAM_NORMAL;
ptr = fgets(datrec, 2048, infile);
if (ptr == NULL)
  if (errno == 0)
    il = FLAM_EOF;
    else
         { rc = errno + FLAM_IO;
           closfiles(rc, infile, &flmid);
         };
while (il == FLAM_NORMAL)
  {
    reclength = strlen(datrec) - 1;
                                 /* Komprimat ausgeben */
    flmput(&flmid, &rc, &reclength, datrec);
    if (rc != FLAM_NORMAL)
      closfiles(rc, infile, &flmid);
                                 /* Originaldatei lesen */
    ptr = fgets(datrec, 2048, infile);
    if (ptr == NULL)
      if (errno == 0)
        il = FLAM_EOF;
        else
            { rc = errno + FLAM_IO;
              closfiles(rc, infile, &flmid);
            };
  };

                                 /* FLAM beenden */
flmcls(&flmid, &rc, &cputime, &zunks, &zunkb, &zunkbofl,
       &zks, &zkb, &zkbofl);
                                 /* Statistik ausgeben */
printf("\nAnzahl unkomprmierte Saetze: %d\n", zunks);
printf("Anzahl unkomprmierte Bytes:  %d\n", zunkb);
printf("\nAnzahl unkomprmierte Saetze: %d\n", zks);
printf("Anzahl unkomprmierte Bytes:  %d\n", zkb);

if (rc != FLAM_NORMAL)
    closinput(rc, infile);

                                 /* Originaldatei schliessen */
rc = fclose(infile);

exit(rc);

}
```

```
/*
      Dateien schliessen, Programm beenden
*/

closfiles(rc, infile, flmid)
unsigned long rc;
FILE *infile;
char **flmid;

{
long il;
unsigned long cputime;           /* CPU-Zeit */
unsigned long zks;               /* Satzzaehler */
unsigned long zkb;               /* Bytezaehler */
unsigned long zkbofl;            /* Bytezaehler Ueberlauf */
unsigned long zunks;             /* Satzzaehler */
unsigned long zunkb;             /* Bytezaehler */
unsigned long zunkbofl;          /* Bytezaehler Ueberlauf */

                                 /* FLAM beenden */
flmcls(flmid, &il, &cputime, &zunks, &zunkb, &zunkbofl,
       &zks, &zkb, &zkbofl);
                                 /* Statistik ausgeben */
if (il == FLAM_NORMAL)
  {
    printf("\nAnzahl unkomprmierte Saetze: %d\n", zunks);
    printf("Anzahl unkomprmierte Bytes:  %d\n", zunkb);
    printf("\nAnzahl unkomprmierte Saetze: %d\n", zks);
    printf("Anzahl unkomprmierte Bytes:  %d\n", zkb);
  };
closinput(rc, infile);

}

/*
Originaldatei schliessen, Programm beenden
*/

closinput(rc, infile)
unsigned long rc;
FILE *infile;

{

long il;
                                 /* Originaldatei schliessen */
il = fclose(infile);
put_flmsg(rc);

exit(rc);

}
```


7.6.2 Dekomprimieren einer Datei

Die von FLAM dekomprimierten Sätze einer Datei werden abgerufen. Der Fileheader, der aus dem allgemeinen und dem anwenderspezifischen Teil besteht, wird gelesen. Die Namen der Ein- und Ausgabedateien werden im Dialog eingegeben.

```
/*

Beispiel: Dekomprimieren einer Datei
File Header mit System- und Anwenderteil wird gelesen

*/

#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
extern int errno;
extern int sys_nerr;

#include "flamincl.h"

main  ()

{

char *flmid;                    /* Kennung Satzschnittstelle */

char kname[100];                /* Name Komprimatsdatei */
char oname[100];                /* Name Originaldatei */
char dname[100];                /* Name dekomprimierte Datei */

long flcode;                    /* Flamcode */
long modus;                     /* Modus der Komprimierung */
long statis;                    /* Statistik */
long opmode;                    /* Openmodus */
long blkmode;                   /* Blockmodus */
long header;                    /* File Header */
long prctrl;                    /* Vorschubsteuerzeichen */
long version;                   /* FLAM Version */
long reclength;                 /* Satzlaenge */
long buflength;                 /* Pufferlaenge */
long namlen;                    /* Laenge Dateiname */
long sattrlen;                  /* Laenge systemspez. Information */

char sysattr[2048];             /* systemspez. Information */
char datrec[2048];              /* Datenbereich */
char system[2];                 /* Betriebssystem */

long sanz;                      /* max. Satzanzahl / Block */
unsigned long maxb;             /* max. Puffergroesse in KB */
long lstpar;                    /* Letzter Parameter */
```

```
                                  /* Komprimatsdatei */
long blksk;                       /* Blocksize */
long recfk;                       /* Satzformat */
long recsk;                       /* Satzlaenge */
long devk;                        /* Geraet */
long orgak;                       /* Organisation */
union { char c[4];
        unsigned long i;
      } recdelk;                  /* Satztrenner */
long cldispk;                     /* Close Disposition */
struct kd keydesck;               /* Schluesselbeschreibung */
                                  /* Dekomprimierte Datei */
long devd;                        /* Geraet */
long orgad;                       /* Organisation */
long blksd;                       /* Blocksize */
long recfd;                       /* Satzformat */
long recsd;                       /* Satzlaenge bei fix */
union { char c[4];
        unsigned long i;
      } recdeld;                  /* Satztrenner */
long cldispd;                     /* Close Disposition */
struct kd keydescd;               /* Schluesselbeschreibung */
                                  /* Originaldatei */
long blkso;                       /* Blocksize */
long recfo;                       /* Satzformat */
long recso;                       /* Satzlaenge */
long devo;                        /* Geraet */
long orgao;                       /* Organisation */
union { char c[4];
        unsigned long i;
      } recdelo;                  /* Satztrenner */
long cldispo;                     /* Close Disposition */
struct kd keydesco;               /* Schluesselbeschreibung */

char exk20[34];                   /* Name Exitroutine */
char exd20[34];                   /* Name Exitroutine */

long il, ir;                      /* Integer */
unsigned long rc;                 /* Return Code */
unsigned long cputime;            /* CPU-Zeit */

unsigned long zks;                /* Satzzaehler */
unsigned long zkb;                /* Bytezaehler */
unsigned long zkbofl;             /* Bytezaehler Ueberlauf */
unsigned long zunks;              /* Satzzaehler */
unsigned long zunkb;              /* Bytezaehler */
unsigned long zunkbofl;           /* Bytezaehler Ueberlauf */

FILE *outfile;                    /* Ausgabedatei */

long uattrlen;
char usrattr[1000];
```

```
                                 /* Dateinamen eingeben */
printf("Dateiname input: ");
scanf("%s", kname);

printf("Dateiname output: ");
scanf("%s", dname);

                                  /* FLAM initialisieren */
opmode = FLAM_C_OPEN_INPUT;       /* dekomprimieren */
statis = FLAM_C_STATISTIC;        /* Statistik */
lstpar = FLAM_C_MORE_PARAMETER;
flmopn(&flmid, &rc, &lstpar, &opmode, kname, &statis);
if (rc != FLAM_NORMAL)
  { put_flmsg(rc);
    exit (rc);
  };

                                  /* Angaben zur Komprimatsdatei */
il = 100;
cldispk = FLAM_C_CLOSE_REWIND;    /* close disposition */
flmopd(&flmid, &rc, &lstpar, &il, kname, &orgak, &recfk, &recsk,
       recdelk.c, &keydesck, &blksk, &cldispk, &devk);
if (rc != FLAM_NORMAL)
  { put_flmsg(rc);
    exit (rc);
  };

                                  /* Angaben zur Komprimierung */
header = FLAM_C_FILEHEADER;       /* File Header lesen */
keydesco.keyparts = 0;
exk20[0] = ' ';                   /* keine Exitroutine */
exd20[0] = ' ';
flmopf(&flmid, &rc, &version, &flcode, &modus, &maxb, &header, &sanz,
       &keydescd, &blkmode, exk20, exd20);
if (rc != FLAM_NORMAL)
  { put_flmsg(rc);
    exit (rc);
  };

                                  /* Fileheader lesen */
                                  /* allgemeine Informationen */
namlen = 100;
flmghd(&flmid, &rc, &namlen, oname, &orgao, &recfo, &recso,
       recdelo.c, &keydesco, &blkso, &prctrl, system);
if ((rc != FLAM_NORMAL) && (rc != FLAM_NO_FILEHEADER))
  closflam(rc, &flmid);

                                  /* System VAX/VMS ? */
if (rc != FLAM_NO_FILEHEADER)
  {
                                  /* anwenderspez. Informationen */
    uattrlen = 1000;
    flmguh(&flmid, &rc, &uattrlen, usrattr);
    if ((rc != FLAM_NORMAL) && (rc != FLAM_NO_FILEHEADER))
      closflam(rc, &flmid);
  };
```

```
                                  /* dekomprimierte Datei oeffnen */

outfile = fopen(dname, "w");
if (outfile == NULL)
  exit(1);

                                  /* Datei dekomprimieren und schreiben */
buflength = 2048;
flmget(&flmid, &rc, &reclength, datrec, &buflength);
if (rc != FLAM_NORMAL)
  {
    if (rc == FLAM_EOF)
      rc = FLAM_NORMAL;
    closfiles(rc, outfile, &flmid);
  };
il = FLAM_NORMAL;
while (il == FLAM_NORMAL)
  {
    datrec[reclength++] = 0x0a;
    datrec[reclength] = 0x00;
    ir = fputs(datrec, outfile);
    if (ir != reclength)
      { rc = FLAM_WRITE_ERR;
        closfiles(rc, outfile, &flmid);
      };
                                  /* dekomprimierten Satz lesen */
    flmget(&flmid, &rc, &reclength, datrec, &buflength);
    if (rc != FLAM_NORMAL)
      {
        if (rc == FLAM_EOF)
         il = rc;
         else
             closfiles(rc, outfile, &flmid);
      };
  };

                                  /* dekomprimierte Datei schliessen */
rc = close(outfile);

                                  /* FLAM beenden */
flmcls(&flmid, &rc, &cputime, &zunks, &zunkb, &zunkbofl, &zks,
       &zkb, &zkbofl);
                                  /* Statistik ausgeben */
printf("\nAnzahl unkomprmierte Saetze: %d\n", zunks);
printf("Anzahl unkomprmierte Bytes:  %d\n", zunkb);
printf("\nAnzahl komprmierte Saetze: %d\n", zks);
printf("Anzahl komprmierte Bytes:  %d\n", zkb);
put_flmsg(rc);
exit (rc);
}
```

```
/*
dekomprimierte Datei schliessen, FLAM beenden
*/

closfiles(rc, outfile, flmid)

unsigned long rc;
FILE *outfile;
char **flmid;

{
long il;
                                  /* dekomprimierte Datei schliessen */
il = close(outfile);

closflam(rc, flmid);

}

/*
FLAM beenden, Programm beenden
*/

closflam(rc, flmid)

unsigned long rc;
char **flmid;

{
long il;
unsigned long cputime;            /* CPU-Zeit */
unsigned long zks;                /* Satzzaehler */
unsigned long zkb;                /* Bytezaehler */
unsigned long zkbofl;             /* Bytezaehler Ueberlauf */
unsigned long zunks;              /* Satzzaehler */
unsigned long zunkb;              /* Bytezaehler */
unsigned long zunkbofl;           /* Bytezaehler Ueberlauf */

                                  /* FLAM beenden */
flmcls(flmid, &il, &cputime, &zunks, &zunkb, &zunkbofl,
       &zks, &zkb, &zkbofl);
                                  /* Statistik ausgeben */
if (il == FLAM_NORMAL)
  {
    printf("\nAnzahl unkomprmierte Saetze: %d\n", zunks);
    printf("Anzahl unkomprmierte Bytes:  %d\n", zunkb);
    printf("\nAnzahl unkomprmierte Saetze: %d\n", zks);
    printf("Anzahl unkomprmierte Bytes:  %d\n", zkb);
  };

if (rc == FLAM_NORMAL)
  rc = il;

                                  /* Return umsetzen */
put_flmsg(rc);
exit (rc);
}
```

7.7 Benutzereigene Ein-/Ausgabe

In den folgenden Beispielen werden die Schnittstellen für die Ein- und Ausgabemodule der Anwender beschrieben.

Angegeben sind zu jedem Modul die Aufrufe mit den erforderlichen Parametern und den möglichen Werten. Ein- und Ausgabefehler sind nicht enthalten.

7.7.1 Öffnen einer Datei

```
/*

Oeffnen einer Datei zum Lesen oder Schreiben (user_io)

*/

#include "flamincl.h"

void usropn(workio, retco, openmode, linkname, fcbtype, recform, recsize,
            blksize, keydesc, device, recdelim, padchar, prcrtl,
            closdisp, access,  namelen, filename)

FLAM_PPCHAR workio;    /*    Kennung der Datei */
FLAM_PLONG *retco;     /*    Returncode */
FLAM_PLONG *openmode;  /*    Verarbeitungsart */
                       /*    0 = input (seq. lesen)
                             1 = output (seq. schreiben)
                             2 = inout (mit Schluessel lesen und
                                 schreiben) (vorhandene Datei)
                             3 = outin (mit Schluessel schreiben
                                 und lesen)
                                 (neue Datei) */
FLAM_PPCHAR linkname;   /*   Linkname / Dateiname */
FLAM_PLONG *fcbtype;    /*   Organisation */
                        /*   0,  8, 16, ... = sequentiell
                             1,  9, 17, ... = indexsequentiell
                             2, 10, 18, ... = relativ
                             3, 11, 19, ... = Direktzugriff
                             5, 13, 21, ... = Bibliothek
                             6, 14, 22, ... = physisch
                                   negativ = systemspezifische
                                             Attribute uebernehmen
                             (bei open = output, sattrlen ungleich 0)  */
FLAM_PLONG *recform;    /*   Satzformat */
                        /*   0, 8, 16, ... = variabel
                             8  = blocked,
                             16 = blocked/spanned
                             24 = VFC
                             32 = 2 Byte Laengenfeld
                             40 = 4 Byte ASCII Längenfeld
                             48 = 4 Byte EBCDIC Längenfeld
```

```
                                 1, 9, 17, ... = fix
                                 8  = blocked,
                                 16 = blocked/standard
                                 2, 10, 18, ... = undefined
                                 18 = EAF
                                 3, 11, 19, ... = Stream
                                 negativ = systemspezifische
                                           Attribute uebernehmen
                                           (bei open = output, sattrlen
                                           ungleich 0)  */
FLAM_PLONG *recsize;     /*     Satzlaenge */
                         /*     0 - 32767
                                 recform v: max. Satzlaenge / 0
                                 recform f: Satzlange
                                 recform u: max. Satzlaenge / 0
                                 recform s: max. Satzlaenge / 0 */
FLAM_PLONG *blksize;     /*     Blocklaenge */
                         /*     0 = ungeblockt
                                 1 - 32767
                                 negativ = systemspezifische
                                           Attribute uebernehmen
                                           (bei open = output, sattrlen
                                           ungleich 0)  */
PSTRKD *keydesc;         /*     Schluesselbeschreibung */
FLAM_PLONG *device;      /*     Geraetetyp */
                         /*     0 = Platte / nicht bekannt
                                 1 = Band
                                 2 = Diskette
                                 3 = Streamer
                                 7/15/23 = User-i/o
                                 negativ = systemspezifische
                                           Attribute uebernehmen
                                           (bei open = output, sattrlen
                                           ungleich 0) */
FLAM_PCHAR recdelim;     /*     Texttrenner, beendet mit '\0'
                                 bei vfc enthaelt das 1. Byte die
                                 Headerlaenge */
FLAM_PCHAR padchar;      /*     Fuellzeichen */
FLAM_PLONG *prcrtl;      /*     Vorschubsteuerzeichen */
FLAM_PLONG *closdisp;    /*     Closeverarbeitung */
                         /*     0 = rewind
                                 1 = unload
                                 2 = retain / leave */
FLAM_PLONG *access;      /*     Zugriffsverfahren */
                         /*     0 = logisch
                                 1 = physisch (blockweise)
                                 2 = mixed (Blockzugriff mit Satzuebergabe) */

FLAM_PLONG *namelen;     /*     Laenge Dateiname (expanded name) */
FLAM_PPCHAR filename;    /*     Dateiname (expanded name) */
```

```
{

*retco = 0;

/*
Datei Oeffnen
*/
.
.
.
ende: ;

return;

}
```

7.7.2 Lesen einer Datei

```
/*

  Datei lesen (user_io)

*/

#include "flamincl.h"

void usrget(workio, retco, reclen, record, buflen)

FLAM_PPCHAR workio;      /*   Kennung der Datei */
FLAM_PLONG *retco;       /*   Returncode */
FLAM_PLONG *reclen;      /*   Satzlaenge in Bytes */
FLAM_PPCHAR record;      /*   Satz */
FLAM_PLONG *buflen;      /*   Laenge des Satzpuffers in Bytes */

{

*retco = 0;

/*
        Satz lesen
*/
.
.
.
ende:

return;

}
```


7.7.3 Schreiben einer Datei

```
/*

  Datei schreiben (user_io)

*/

#include "flamincl.h"

void usrput(workio, retco, reclen, record)

FLAM_PPCHAR workio;     /*   Kennung der Datei */
FLAM_PLONG *retco;      /*   Returncode */
FLAM_PLONG *reclen;     /*   Satzlaenge in Bytes */
FLAM_PPCHAR record;     /*   Satz */

{

*retco = 0;

/*
        Satz schreiben
*/
.
.
.
ende:

return;

}
```


7.7.4 Schließen einer Datei

```
/*

  Schliessen einer Datei (user_io)

*/

#include "flamincl.h"

void usrcls(workio, retco)

FLAM_PPCHAR workio;     /*   Kennung der Datei */
FLAM_PLONG *retco;      /*   Returncode */

{

*retco = 0;

/*
        Schliessen einer Datei
*/
.
.
.
ende:

return;

}
```


7.8 Die Benutzerausgänge

7.8.1 Mit exk10 Sätze einer bestimmten Satzart aus einer Datei selektieren und komprimieren

Von FLAM werden die Sätze einer Datei gelesen und an exk10 zur Bearbeitung übergeben. In exk10 wird die Satzart (1. Zeichen im Satz) geprüft. Sätze mit Satzart "1" werden an FLAM zur Komprimierung zurückgegeben (returncode=0), alle anderen werden nicht weiterbearbeitet (returncode=4).

```
/*

       Mit exitk10 Saetze mit einer bestimmten Satzart
       aus einer Datei selektieren und komprimieren

*/

exk10(functioncode, returncode, record_pointer, record_length, work)

long *functioncode;          /* Funktionscode */
long *returncode;            /* Returncode */
char **record_pointer;       /* Satzpointer */
long *record_length;         /* Satzlaenge */
char *work;                  /* Arbeitsbereich */

{

*returncode = 0;

                             /* Aufruf nach open oder vor close */
if ((*functioncode == 0) ||  (*functioncode == 8))
  return;

if (**record_pointer != '1')
                             /* Saetze mit Satzart 1 (1. Zeichen
                                im Satz) werdenkomprimiert, alle
                                anderen Saetze werden ueberlesen */
  *returncode = 4;

return;

}
```

7.8.2 Mit exd10 Sätze einer komprimierten Datei verarbeiten

Die dekomprimierten Sätze werden nicht in die dekomprimierte Datei geschrieben, sondern von FLAM an exd10 zur Verarbeitung übergeben. exd10 verarbeitet die Sätze (hier Anzeige am Bildschirm) und gibt sie nicht zum Schreiben zurück (returncode=4).

```
/*

       Mit exitd10 Saetze einer komprimierten Datei verarbeiten.
       Die dekomprimierten Saetze werden nicht in die dekomprimierte
       Datei geschrieben.

*/

#include  <stdio.h>

exd10(functioncode, returncode, record_pointer, record_length, work)

long *functioncode;           /* Funktionscode */
long *returncode;             /* Returncode */
char **record_pointer;        /* Satzpointer */
long *record_length;          /* Satzlaenge */
char *work;                   /* Arbeitsbereich */

{

*returncode = 0;

if ((*functioncode == 0) ||  (*functioncode == 8))
  return;


/*

Verarbeitung (Anzeigen) des Dateisatzes

*/

*(*record_pointer + *record_length) = 0x00;
*work = printf("%s\n", *record_pointer);
*returncode = 4;              /* kein Fehler, Satz nicht uebernehmen */

return;

}
```

7.8.3 exk20 - Vertauschen von 2 Byte bei der Komprimierung

Um die komprimierte Datei zusätzlich zu schützen, werden in jedem Satz 2 Byte vertauscht. Würden die Daten dekomprimiert ohne die Bytes wieder zurückzutauschen, würde dabei ein Fehler auftreten.

Bei der Dekomprimierung werden mit dem gleichen Programm die vertauschten Bytes an ihre ursprüngliche Stelle zurückgesetzt.

```
/*

       Beispiel fuer exitk20 (und exitd20)
       Vertauschen von 2 Byte bei der Komprimierung.

       Bei der Dekomprimierung werden mit dem gleichen Programm
       die vertauschten  Bytes an ihre urspruengliche Stelle
       zurueckgesetzt.
       Der Programmaufruf heisst dann exd20 statt exk20.

*/

exk20(functioncode, returncode, record_pointer, record_length, work)

long *functioncode;         /* Funktionscode */
long *returncode;           /* Returncode */
char **record_pointer;      /* Satzpointer */
long *record_length;        /* Satzlaenge */
char *work;                 /* Arbeitsbereich */

{

*returncode = 0;            /* kein Fehler, Satz uebernehmen */

if ((*functioncode == 0) ||  (*functioncode == 8))
  return;

if (*record_length > 16)    /* 16. und 17. Byte vertauschen */
  { *work = *(*record_pointer + 15);
    *(*record_pointer + 15) = *(*record_pointer + 16);
    *(*record_pointer + 16) = *work;
  };

return;

}
```

7.8.4 exd20 - Vertauschen von 2 Byte bei der Dekomprimierung

Die bei der Komprimierung mit exk20 vertauschten Bytes werden wieder zurückgetauscht (siehe Abschnitt 7.8.3).

Gleiches Programm wie exitk20 bei der Komprimierung.

```
/*

     Beispiel fuer exitd20
     Vertauschen von 2 Byte bei der Dekomprimierung.

     Gleiches Programm wie exitk20 bei der Komprimierung.

*/

exd20(functioncode, returncode, record_pointer, record_length, work)

long *functioncode;         /* Funktionscode */
long *returncode;           /* Returncode */
char **record_pointer;      /* Satzpointer */
long *record_length;        /* Satzlaenge */
char *work;                 /* Arbeitsbereich */

{

*returncode = 0;            /* kein Fehler, Satz uebernehmen */

if ((*functioncode == 0) || (*functioncode == 8))
  return;

if (*record_length > 16)    /* 16. und 17. Byte vertauschen */
  { *work = *(*record_pointer + 15);
    *(*record_pointer + 15) = *(*record_pointer + 16);
    *(*record_pointer + 16) = *work;
  };

return;

}
```

7.8.5 Funktion zur automatischen Schlüsselverwaltung

Um bei der Verschlüsselung/Entschlüsselung Kennwörter automatisch zu erzeugen bzw. zu übergeben, kann beim Aufruf von FLAM über den -kmexit-Parameter eine entsprechende Funktion aktiviert werden. Eine solche Funktion könnte etwa (als ANSI-C-Programm) wie folgt aussehen:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>

#define  KMX_KEY_LENGTH        8
#define  KMX_ID_INFO           0xFFFFFFFF
#define  KMX_ENCRYPT           1
#define  KMX_DECRYPT           0
void samplex1
/***********************************************/
(                          /* To generate a password (fuco = 1), this exit */
  signed   long *fuco,     /* routine randomly positions into a text file  */
  signed   long *retco,    /* and extracts a string of length 8. It passes */
  unsigned long *parmlen,  /* back the string in cryptokey and the blurred */
  unsigned char *param,    /* random position in data.                     */
  unsigned long *datalen,  /* To retrieve a password (fuco = 0), the read  */
  unsigned char *data,     /* position is derived from data and the string */
  unsigned long *ckylen,   /* found there is passed back in cryptokey.     */
  unsigned char *cryptokey,/* With fuco = -1, only a text line identifying */
  unsigned long *msglen,   /* the routine is passed back in message.       */
  unsigned char *message
/***********************************************/
)
{ const char          exit_id[] = "Key Management Exit Version 1.0.0";
  const char          ffkmx  [] = "/flam/ffkmx";
  const char          errmsg1[] = "Can't get info on file /flam/ffkmx";
  const char          errmsg2[] = "Can't open file /flam/ffkmx";
  const char          errmsg3[] = "Positioning in file /flam/ffkmx failed";
  const char          errmsg4[] = "Reading file /flam/ffkmx failed";
  const char          errmsg5[] = "Function code %d not supported";
  const char          errmsg6[] = "File /flam/ffkmx too small";
  unsigned long       count1   ;
  unsigned long       count2   ;
  unsigned long       rnd_nbr  ;
  unsigned long       read_pos ;
  FILE                *khandle ;
  time_t              unixtime ;
  struct tm           *p_tm    ;
  struct stat         filestat ;

  *msglen = 0;                          /*                         */
                                        /*                         */
  *retco  = stat(ffkmx, &filestat);     /* Determine file size     */
  if (*retco != 0)                      /*                         */
  { strcpy(message, (char*)errmsg1);    /*                         */
    *msglen = strlen(message);          /*                         */
    return;                             /*                         */
  }                                     /*                         */
```

```
if (filestat.st_size < 256)             /* Check file's size       */
{ strcpy(message, (char*)errmsg6);      /*                         */
  *msglen = strlen(message);            /*                         */
  return;                               /*                         */
}                                       /*                         */
                                        /*                         */
khandle = fopen(ffkmx, "rb");           /* Open text file          */
if (khandle == NULL)                    /*                         */
{ strcpy(message, (char*)errmsg2);      /*                         */
  *msglen = strlen(message);            /*                         */
  return;                               /*                         */
}                                       /*                         */
                                        /*                         */
switch (*fuco)                          /*                         */
{ case KMX_ENCRYPT  :                   /* For Encryption:         */
         time(&unixtime);               /*                         */
         srandom(unixtime % 0x00010001L);/*   generate random #     */
         read_pos = rnd_nbr             /*                         */
                  = random() % filestat.st_size;/* as read position and */
         rnd_nbr  ^= 0x55555555L;           /*   obscure it             */
         data[3]   = (char)rnd_nbr;         /*   save modified #        */
         rnd_nbr >>= 8;                     /*                          */
         data[2]   = (char)rnd_nbr;         /*                          */
         rnd_nbr >>= 8;                     /*                          */
         data[1]   = (char)rnd_nbr;         /*                          */
         rnd_nbr >>= 8;                     /*                          */
         data[0]   = (char)rnd_nbr;         /*                          */
         *datalen  = 4;                     /*                          */
         break;                             /*                          */
                                            /*                          */
  case KMX_DECRYPT:                         /* For Decryption           */
         rnd_nbr   = (unsigned long)data[0]; /*   retrieve saved number */
         rnd_nbr <<= 8;                     /*                          */
         rnd_nbr  += (unsigned long)data[1]; /*                         */
         rnd_nbr <<= 8;                     /*                          */
         rnd_nbr  += (unsigned long)data[2]; /*                         */
         rnd_nbr <<= 8;                     /*                          */
         rnd_nbr  += (unsigned long)data[3]; /*                         */
         read_pos  = rnd_nbr ^ 0x55555555L;  /*   undo obscuring        */
                                            /*                          */
         break;                             /*                          */
                                            /*                          */
  case KMX_ID_INFO:                         /* For Info request         */
         strcpy(message, (char*)exit_id);   /*   pass ID_string         */
         *msglen = strlen(message);         /*                          */
         *retco = 0;                        /*                          */
         return;                            /*                          */
                                            /*                          */
  default          :                        /* For all other codes      */
         sprintf(message, errmsg5, *fuco);  /*   issue error            */
         *msglen = strlen(message);         /*                          */
         *retco = -1;                       /*                          */
         return;                            /*                          */
}                                           /*                          */
*retco  = fseek(khandle, read_pos, SEEK_SET);/* Position on read pos.   */
if (*retco != 0)                            /*                          */
{ strcpy(message, (char*)errmsg3);          /*                          */
  *msglen = strlen(message);                /*                          */
  return;                                   /*                          */
}                                           /*                          */
```

```
  count1 = filestat.st_size - read_pos;        /* Check if file wrap      */
  if (count1 >= KMX_KEY_LENGTH)                /* necessary               */
  { *retco = fread(cryptokey, 1,               /* if not, read full key   */
                   KMX_KEY_LENGTH, khandle);   /*                         */
    if (*retco != KMX_KEY_LENGTH)              /*                         */
    { strcpy(message, (char*)errmsg4);         /*                         */
      *msglen = strlen(message);               /*                         */
      *retco = -1;                             /*                         */
      return;                                  /*                         */
    }                                          /*                         */
    *retco = 0;                                /*                         */
  }                                            /*                         */
  else                                         /*                         */
  { *retco = fread(cryptokey, 1, count1, khandle);/* if yes,read first part */
    if (*retco != count1)                      /*                         */
    { strcpy(message, (char*)errmsg4);         /*                         */
      *msglen = strlen(message);               /*                         */
      *retco = -1;                             /*                         */
      return;                                  /*                         */
    }                                          /*                         */
    read_pos = 0;                              /*                         */
    *retco  = fseek(khandle, read_pos, SEEK_SET);/* reposition on beginning */
    if (*retco != 0)                           /* (wrap around)           */
    { strcpy(message, (char*)errmsg3);         /*                         */
      *msglen = strlen(message);               /*                         */
      return;                                  /*                         */
    }                                          /*                         */
    count2 = KMX_KEY_LENGTH - count1;          /*                         */
    *retco = fread(&cryptokey[count1], 1,      /* then read second part   */
                   count2, khandle);           /*                         */
    if (*retco != count2)                      /*                         */
    { strcpy(message, (char*)errmsg4);         /*                         */
      *msglen = strlen(message);               /*                         */
      *retco = -1;                             /*                         */
      return;                                  /*                         */
    }                                          /*                         */
  }                                            /*                         */
  cryptokey[KMX_KEY_LENGTH] = '\0';            /* Terminate string        */
  *ckylen                   = KMX_KEY_LENGTH;  /* Set length              */
  fclose(khandle);                             /* Close file              */
  return;                                      /*                         */
}                                              /*                         */
```

Diese Funktion bedient die Schnittstelle des Benutzerausgangs für automatische Schlüsselverwaltung. Damit FLAM sie aktivieren kann, muss sie Bestandteil einer Shared Library sein, die im lib-Unterverzeichnis des Installationsverzeichnisses von FLAM residiert. Hat diese Shared Library beispielsweise den Namen libuserex.so, so muss, um die Schlüssel über diese Funktion automatisch zu verwalten, im flam-Kommando oder dem Kommandostring an flamup

```
-kmexit=samplex1(libuserex)
```

angegeben werden. Bei der Verschlüsselung (Komprimierung) wird dann ein Schlüssel automatisch erzeugt und an FLAM übergeben. Auch die für die Wiedergewinnung des Schlüssels benötigte Information übergibt die Funktion an FLAM, das sie in einem anwenderspezifischen Fileheader speichert. Bei der

Enschlüsselung (Dekomprimierung) reicht FLAM diese Information zurück an die Funktion und erhält von ihr wieder den passenden Schlüssel.

FLAM (UNIX)

Benutzerhandbuch

Anhang A:

Returncodes

A. Returncodes

Returncode	Symbolischer Name - Bedeutung
-1	FLAM_FCT_NOT_EXIST **Bedeutung:** Funktion nicht verfügbar/vorhanden/erlaubt (Wird von flamup in 999 abgeändert)
0	FLAM_NORMAL **Bedeutung:** Kein Fehler
1	FLAM_RECORD_SHORTENED **Bedeutung:** Dekomprimierter Satz verkürzt (Warnung aus Satzschnittstelle)
2	FLAM_EOF **Bedeutung:** end-of-file
4	FLAM_RECORD_LENGTHENED **Bedeutung:** Dekomprimierte/r Sätze/Satz verlängert
6	FLAM_NEW_FILE **Bedeutung:** Neue Datei beginnt
7	FLAM_NO_PW **Bedeutung:** Kennwort muss angegeben werden
9	FLAM_NO_FILEHEADER **Bedeutung:** Kein Fileheader vorhanden
10	FLAM_NOT_FLAMFILE **Bedeutung:** Datei ist keine FLAMFILE

Returncode	Symbolischer Name - Bedeutung
11	FLAM_FLAMFILE_FORMAT_ERROR **Bedeutung:** FLAMFILE Formatfehler
12	FLAM_FLAMFILE_RECORD_SIZE **Bedeutung:** Komprimats-Satzlänge ungültig
13	FLAM_FILE_LENGTH **Bedeutung:** FLAMFILE unvollständig bzw. beschädigt
14	FLAM_CHECKSUM **Bedeutung:** Checksummenfehler
15	FLAM_RECORD_GR_32KB **Bedeutung:** Originalsatz ist grösser als 32764 Byte
16	FLAM_RECORD_GR_BUFFER **Bedeutung:** Originalsatz ist grösser als Matrix - 4
17	FLAM_FLAM_VERSION_1 **Bedeutung:** FLAM-Komprimat Version 1
20	FLAM_ILLEGAL_FLAM_OPEN **Bedeutung:** Unzulässiger Open Modus
22	FLAM_ILLEGAL_COMPRESSION_MODE **Bedeutung:** Unzulässiges Kompressionsverfahren
25	FLAM_ILLEGAL_RECORD_SIZE **Bedeutung:** Unzulässige Satzlänge
26	FLAM_ILLEGAL_CHARACTER_SET **Bedeutung:** Unzulässiger Zeichencode

Returncode	Symbolischer Name - Bedeutung
29	FLAM_ERR_PW **Bedeutung:** Kennwort fehlt oder falsch angegeben
30	FLAM_FILE_EMPTY **Bedeutung:** Eingabedatei ist leer
31	FLAM_FILE_NOT_FOUND **Bedeutung:** Eingabedatei ist nicht vorhanden
32	FLAM_OPEN_MODE **Bedeutung:** Ungültiger Open Modus
34	FLAM_RECORD_FORMAT **Bedeutung:** Ungültiges Satzformat
35	FLAM_RECORD_SIZE **Bedeutung:** Ungültige Satzlänge
39	FLAM_NO_VALID_FILENAME **Bedeutung:** Kein gültiger Dateiname
43	FLAM_ERR_EXIT **Bedeutung:** Fehlerabbruch durch Exit-Routine
46	FLAM_OPEN_MSGFILE **Bedeutung:** Meldungsdatei kann nicht geöffnet werden
47	FLAM_ERR_OUTPATH **Bedeutung:** -outpath nur mit [ ], [*] oder [dateiname] erlaubt
53	FLAM_PATH_INVALID **Bedeutung:** Umgebungsvariable FLAM_PATH ist nicht definiert

Returncode	Symbolischer Name - Bedeutung
56	FLAM_FILEHEADER_GR_32KB **Bedeutung:** Länge Fileheader im Komprimat 32 KB
57	FLAM_KOMP_LENGTH **Bedeutung:** Länge des Komprimates falsch
59	FLAM_ERR_NONDC **Bedeutung:** ndc nur mit adc, aes oder flamenc zulaessig
60	FLAM_SYNTAX **Bedeutung:** Komprimat fehlerhaft
65	FLAM_CONS_RECORD **Bedeutung:** Konsistenzpunkt falsch
70	FLAM_NOT_VALID_VERSION **Bedeutung:** FLAM-Version der Komprimatsdatei nicht 2x
71	FLAM_FL_CUT **Bedeutung:** Dekomprimierter Satz verkürzt (Fehler aus flamup)
72	FLAM_RECORD_CUT **Bedeutung:** Dekomprimierte/r Sätze/Satz verkürzt (Warnung aus flamup)
73	FLAM_ERR_KOMP_LENGTH **Bedeutung:** Länge des Komprimates falsch
74	FLAM_CHECK_CHARACTER **Bedeutung:** Prüfzeichenfehler
81	FLAM_PARAM_ERR

Bedeutung: Unbekannter Parameter/Qualifizierer

Returncode	Symbolischer Name - Bedeutung
82	FLAM_PARAMETER_VALUE **Bedeutung:** Ungültiger Parameterwert
85	FLAM_SAME_NAME **Bedeutung:** Gleicher Name für Eingabe- und Ausgabedatei
87	FLAM_NO_INPUT_FILE **Bedeutung:** Kein Name für Eingabedatei
88	FLAM_NO_OUTPUT_FILE **Bedeutung:** Kein Name für Ausgabedatei
90	FLAM_QUALIFIER_VALUE **Bedeutung:** Ungültiger Wert eines Qualifizierers
92	FLAM_NO_REPLACING_CHARACTERS **Bedeutung:** Name der Eingabedatei enthält keine zu ersetzenden Zeichen
98	FLAM_READ_ERROR **Bedeutung:** Beim Lesen eines Satzes ist ein Fehler aufgetreten
99	FLAM_WRITE_ERROR **Bedeutung:** Beim Schreiben eines Satzes ist ein Fehler aufgetreten
101	FLAM_NOT_ALL_INPUT_FILES **Bedeutung:** Nicht alle Eingabedateien wurden bearbeitet.
102	FLAM_ANZ_OUTFILE_GR_INFILE **Bedeutung:** Die Anzahl der Ausgabedateien sind größer als die Anzahl Eingabedateien

Returncode	Symbolischer Name - Bedeutung
271	FLAM_ERR_BAD_KMEXIT **Bedeutung:** Fehlerhafte Angabe des Key Management Exits
272	FLAM_ERR_KMEX1 **Bedeutung:** Fehler beim Laden des Key Management Exits
351	FLAM_SEC_ERR_351 **Bedeutung:** SECURITY: Membernummer nicht fortlaufend
352	FLAM_SEC_ERR_352 **Bedeutung:** SECURITY: Membertrailer: Zaehler falsch
354	AES_MEM_MAC **Bedeutung:** AES: Member MACs falsch
355	AES_FIL_MAC **Bedeutung:** AES: File MACs falsch
356	AES_CRC_SUFF **Bedeutung:** Checksummenfehler im Komprimat (CRC-Offs)
357	AES_CRC_5 **Bedeutung:** Checksummenfehler im Komprimat (CRC-5)
360	FLAM_ERR_REDUNDANT **Bedeutung:** Parameter redundant oder nicht eindeutig
401	FLAM_SEQ_OPD **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmopd.

Returncode	Symbolischer Name - Bedeutung
402	FLAM_SEQ_OPF **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmopf.
403	FLAM_SEQ_PHD **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmphd.
404	FLAM_SEQ_GHD **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmghd.
405	FLAM_SEQ_PUH **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmpuh.
406	FLAM_SEQ_GUH **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmguh.
407	FLAM_SEQ_PUT **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmput
408	FLAM_SEQ_GET **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmget
409	FLAM_SEQ_POS **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmpos
410	FLAM_SEQ_FLU **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmflu
411	FLAM_SEQ_CLS **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmcls
412	FLAM_SEQ_PWD **Bedeutung:** Falsche Reihenfolge bei Aufruf von flmpwd

Returncode	Symbolischer Name - Bedeutung
900	FLAM_PARAMETER_QUALIFIER **Bedeutung:** Fehlerhafte FLAM-Optionen (nur flamup)
998	FLAM_INVALID_LICENSE **Bedeutung:** Ungültige Lizenz
999	FLAM_INVALID_FUNCTION **Bedeutung:** Die vorgeschriebene Aufrufsequenz wurde nicht beachtet, oder eine Speicheranforderung konnte nicht befriedigt werden.

Kennzeichnung der I/O-Fehler im höchstwertigen Byte des Returncodes:

1. Halbbyte	Kennzeichen	Betroffene Datei
	X'ex'	Eingabedatei
	X'ax'	Ausgabedatei
	X'fx'	FLAMFILE
	X'cx'	Parameterdatei
	X'dx'	Meldungsdatei
	X'9x'	Codetabelle
	X'8x'	Defaultwerte
	X'7x'	Meldungen

2. Halbbyte	Kennzeichen	Fehlerursprung
	X'x0'	Meldung von FLAM
	X'xf'	Meldung von I-O-Funktionen

FLAM (UNIX)

Benutzerhandbuch

Anhang B:

Codetabellen

B. Codetabellen

Wenn Sie Dateien zwischen Systemen mit unterschiedlichen Zeichencodes austauschen, können Sie im Bedarfsfall mit dem Parameter translate=Codetabelle Ihre Daten durch FLAM automatisch in den gewünschten Zeichencode übersetzen lassen. Hierzu müssen Sie jedoch eine Übersetzungstabelle zur Verfügung stellen.

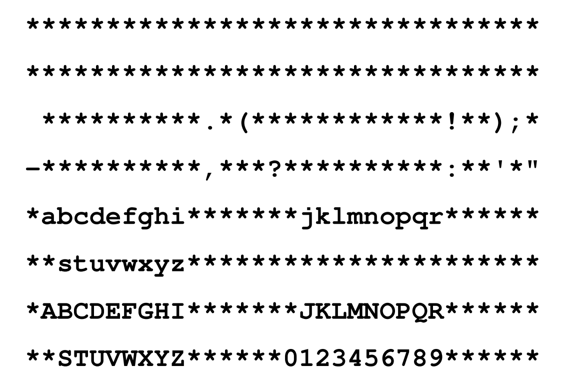
Die folgenden Abschnitte beschreiben, wie Sie eigene Codetabellen erzeugen können.

B.1 Struktur von Codetabellen

FLAM liest aus der Datei, die Sie als Codetabelle spezifiziert haben, die ersten 256 Bytes. Diese 256 Bytes bilden eine Zeichenkette, die von FLAM als Übersetzungstabelle verwendet wird. Dazu wird für jedes Zeichen der Ursprungsdaten der Zahlenwert seines Codes als Index für die Übersetzungstabelle genommen und durch das zugehörige Tabellenelement ersetzt. Der mögliche Wertebereich dieses Zahlenwerts ist 0 - 255 (dezimal) und entspricht den Tabellenpositionen 1 - 256.

Um beispielsweise EBCDIC-codierte Textdaten bei der Dekomprimierung in ASCII-Code zu übersetzen, kann folgende Codetabelle erstellt werden:

Beispiel B.1: Inhalt einer Codetabelle



```
********************************
********************************
 **********.*(************!**);*
-**********,***?**********:**'*"
*abcdefghi*******jklmnopqr******
**stuvwxyz**********************
*ABCDEFGHI*******JKLMNOPQR******
**STUVWXYZ******0123456789******
```

Mit ihr können EBCDIC-Daten übersetzt werden, die alphanumerische Zeichen oder nachstehende Sonderzeichen enthalten: . (!) ; - , ? : ' ". Jeder andere EBCDIC-Code wird in das ASCII-Zeichen * übersetzt.

Die Position jedes Zeichens in der Übersetzungstabelle ergibt sich durch Addition von 1 zum Zahlenwert seines EBCDIC-Codes. So hat etwa die Ziffer 9 den EBCDIC-Code hexadezimal F9 (dezimal 249) und steht an Stelle 250. Sollte beispielsweise die Tabelle um den Umlaut ü erweitert werden, müsste dessen EBCDIC-Code bestimmt werden. Ist dieser Code hexadezimal D0 (dezimal 208), so wäre auf Position 209 (unmittelbar vor dem A) der ASCII-Code für * durch den ASCII-Code für ü zu ersetzen.

Umgekehrt müsste in einer Übersetzungstabelle von ASCII nach EBCDIC für die Ziffer 9 (ASCII-Code hexadezimal 39 = dezimal 57) an 58. Stelle der EBCDIC-Code F9 (hexadezimal) stehen.

B.2 Erstellung von Codetabellen

Beim Erstellen von Codetabellen können Zeichencodes, die nicht druckbaren ASCII-Zeichen entsprechen, im allgemeinen nicht über die Tastatur eingegeben werden. Es empfiehlt sich, folgendes Program mit modifizierter Codetabelle zu verwenden. (siehe Beispiel in /usr/lib/flame/samplest/cr_code.c):

Beispiel B.2: Übersetzungstabelle ASCII - EBCDIC

```
/*
Beispiel:
Übersetzungstabelle ASCII - EBCDIC
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
extern int errno;
extern int sys_nerr;

main()

{ int status;
  unsigned long fk;
  static char filename[20] = {"code_ae.dat"};

  static char ebcdic[256] = {0x00, 0x01, 0x02, 0x03, 0x37, 0x2d, 0x2e, 0x2f,
                             0x16, 0x05, 0x25, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                             0x10, 0x11, 0x12, 0x13, 0x3c, 0x3d, 0x32, 0x26,
                             0x18, 0x19, 0x3f, 0x27, 0x1c, 0x1d, 0x1e, 0x1f,
                             0x40, 0x5a, 0x7f, 0x7b, 0x5b, 0x6c, 0x50, 0x7d,
                             0x4d, 0x5d, 0x5c, 0x4e, 0x6b, 0x60, 0x4b, 0x61,
                             0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
                             0xf8, 0xf9, 0x7a, 0x5e, 0x4c, 0x7e, 0x6e, 0x6f,
                             0x7c, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
                             0xc8, 0xc9, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6,
                             0xd7, 0xd8, 0xd9, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6,
                             0xe7, 0xe8, 0xe9, 0xbb, 0xbc, 0xbd, 0x6a, 0x6d,
                             0x4a, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
                             0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96,
                             0x97, 0x98, 0x99, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6,
                             0xa7, 0xa8, 0xa9, 0xfb, 0x4f, 0xfd, 0xff, 0x07,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0xff, 0x06, 0x01, 0x00
                            };
```

```
  fk = open(filename, O_CREAT | O_WRONLY | O_TRUNC);
  if (fk == -1)
  { perror("Open File");
    exit(1);
  }
  status = chmod(filename, S_IWUSR | S_IRUSR| S_IRGRP | S_IROTH);
  if (status == -1)
  { perror("chmod");
    exit(1);
  }

  status = write(fk, ebcdic, 256);
  if (status != 256)
    perror("Schreibfehler");

  status = close(fk);

  exit(0);
}
```

